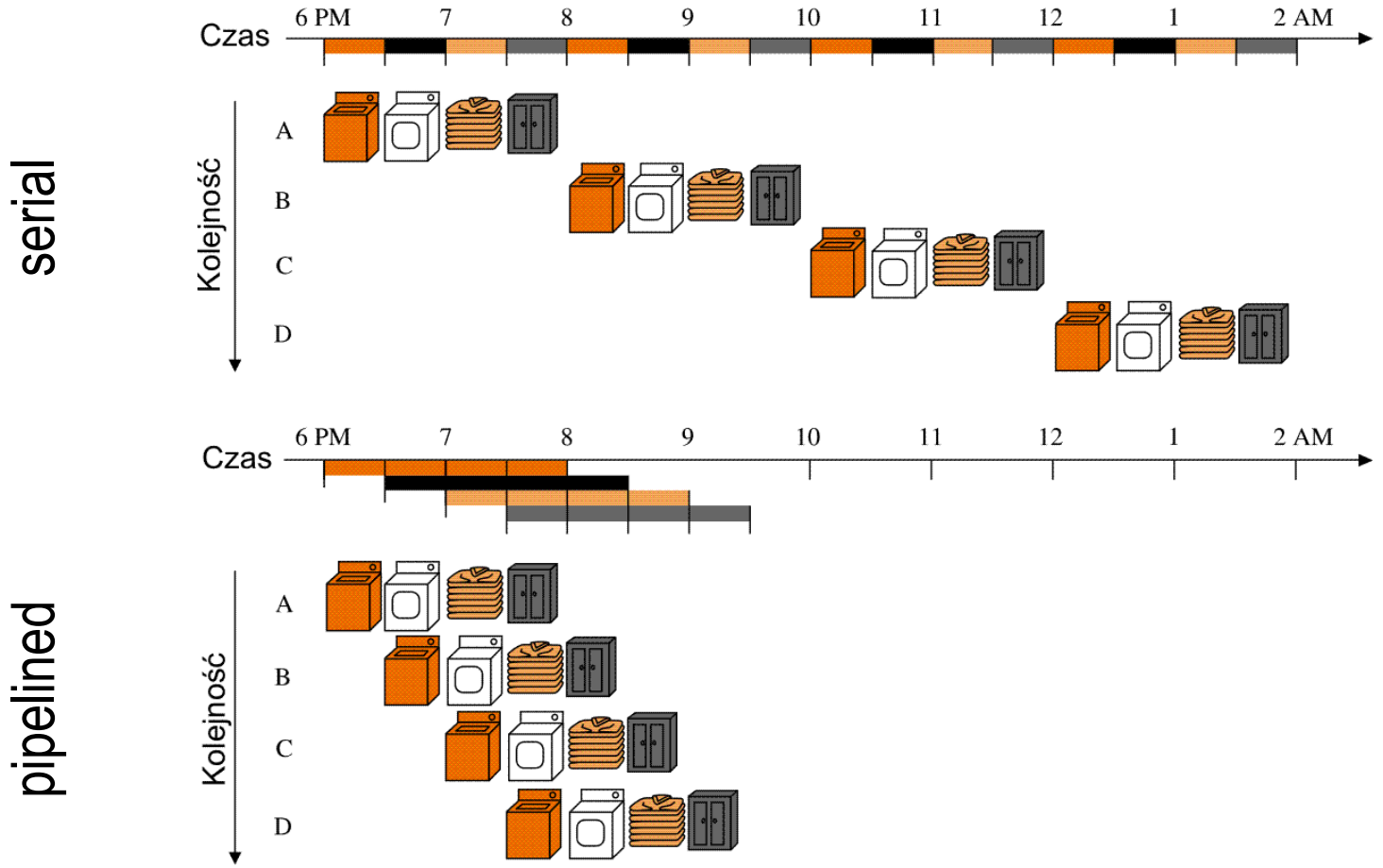
Pipeline Architecture

RISC

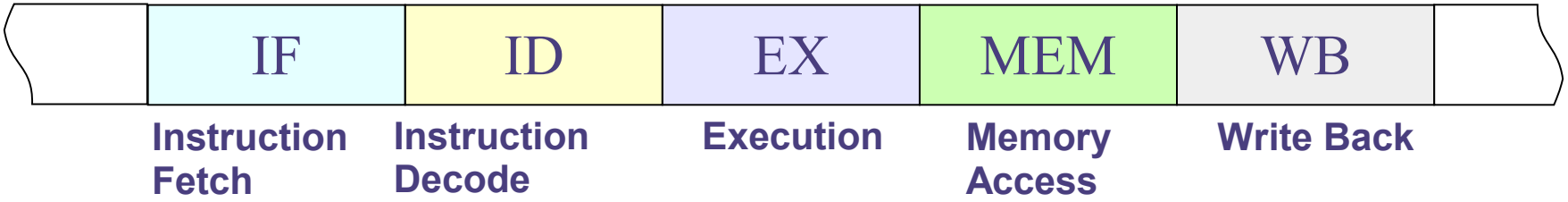
Pipelined vs Serial Processing

- Independent tasks with independent hardware
- No repetitions during the process

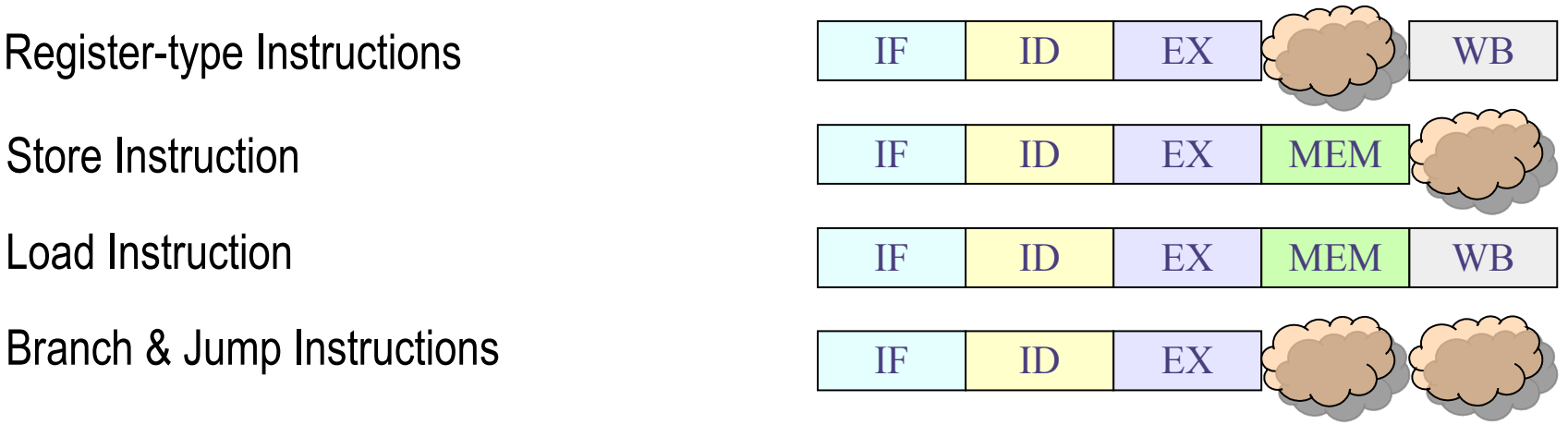


Instruction Machine Cycle

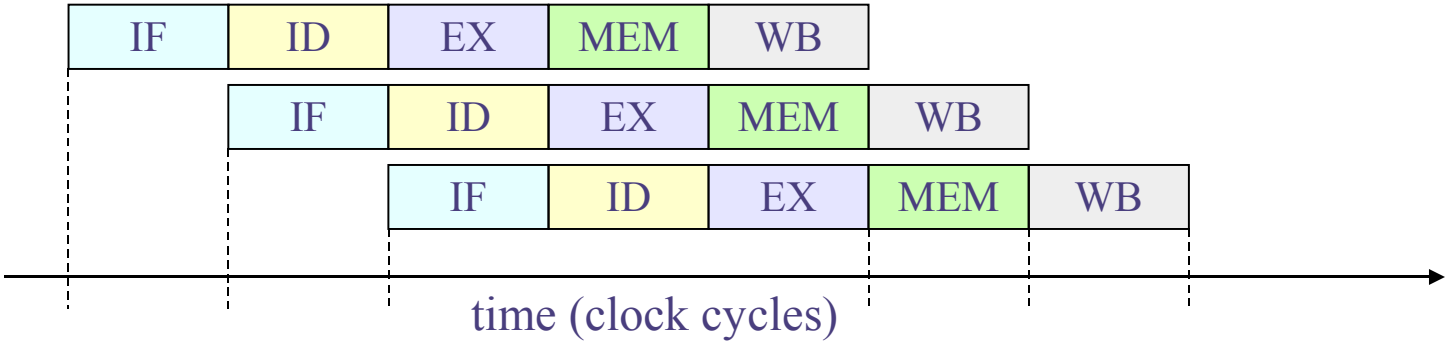
- Every instruction must got through the same stages



- Instruction may not require all the stages, but must execute the all, with empty clock cycles if needed

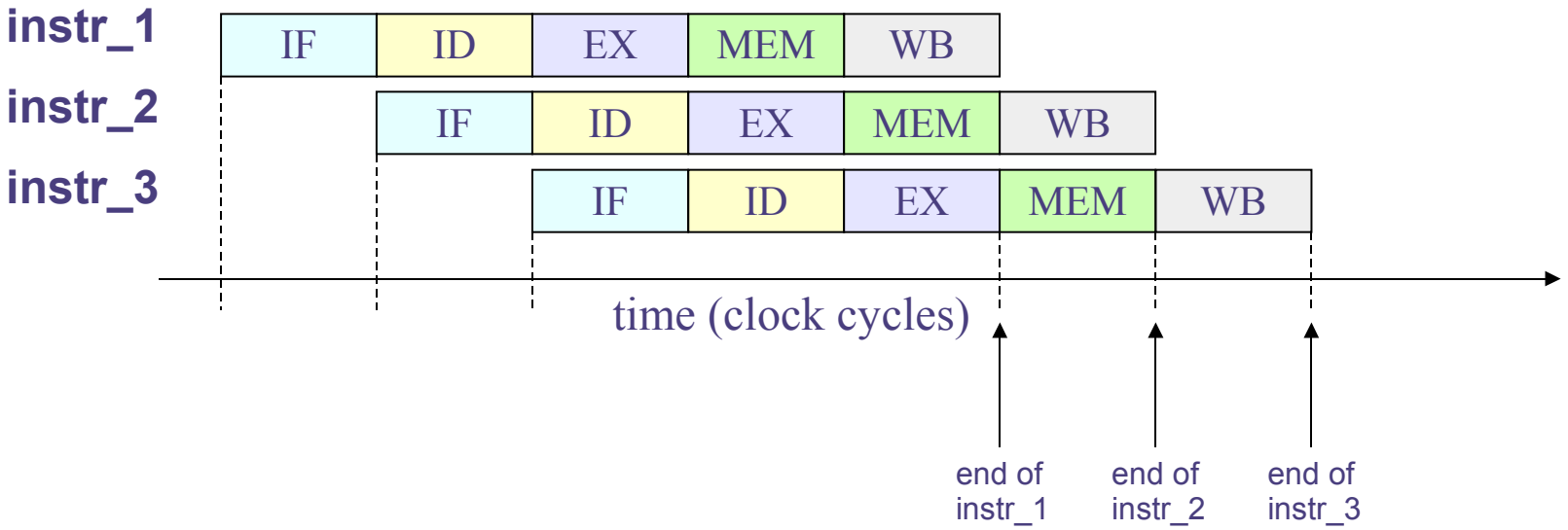


Hardware Resource Usage



- Independent resources in all stages of execution
  - IF – writing IR, Incrementing PC,
  - ID – register decoding
  - EX – operation with ALU
  - MEM – reading or writing the memory
  - WB – writing final result to register file
- At any moment all resources are being used
- At any moment several instructions are processed

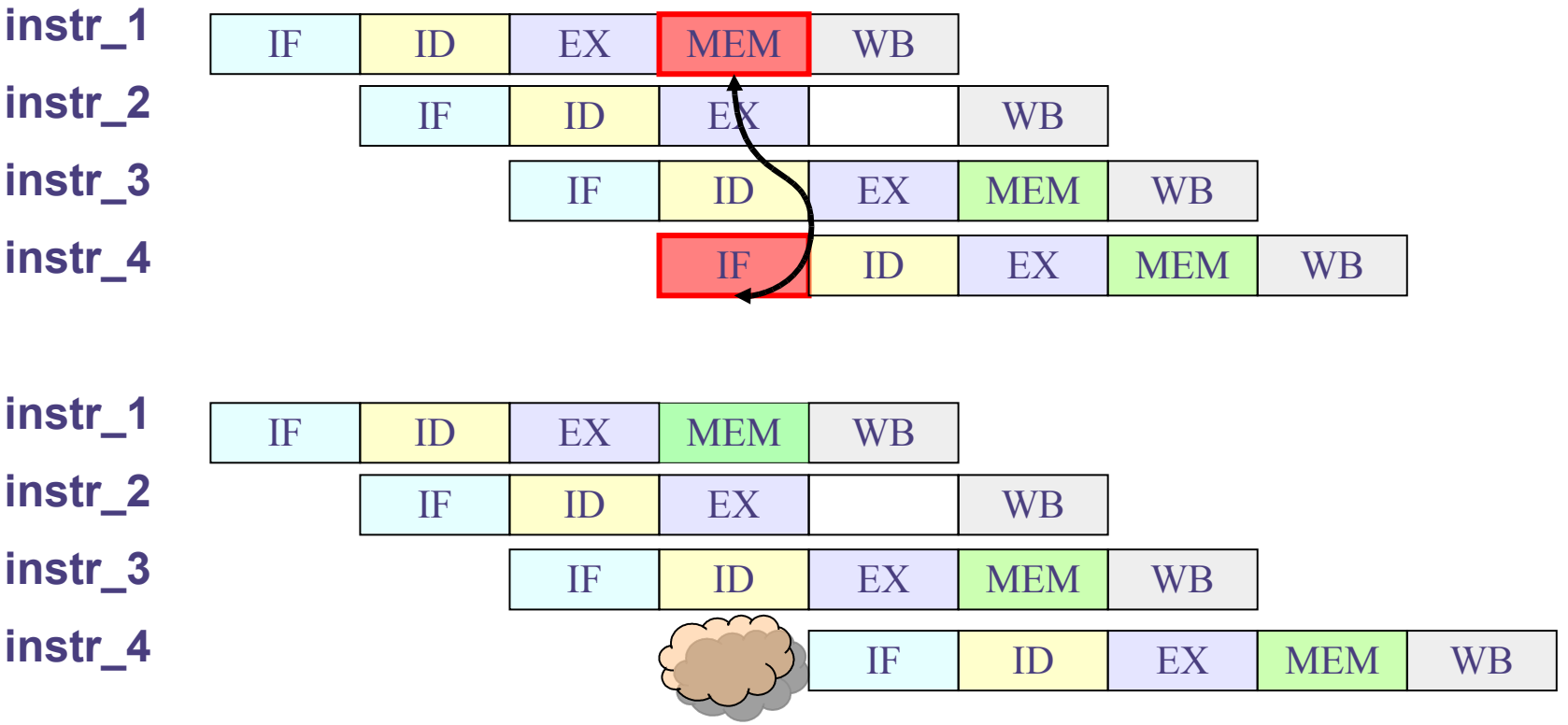
Pipelined Processing Performance



- Every clock cycle one instruction is finished (effectively)
- Speed-up (compared to serial processing) is proportional to the number of stages in machine cycle
  - (e.g. 5 staged = 5 times faster)

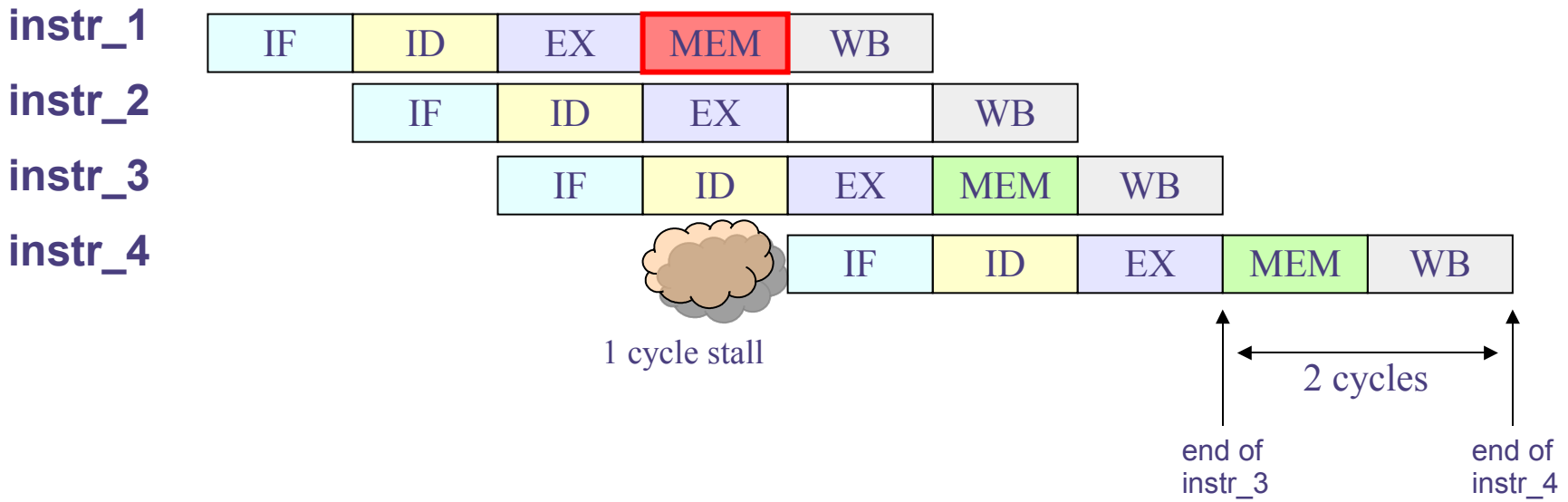
Structural Hazard (Resource Conflict)

- Competition for hardware resources between two instructions in a pipeline
  - (e.g. access to the same memory: IF and MEM)



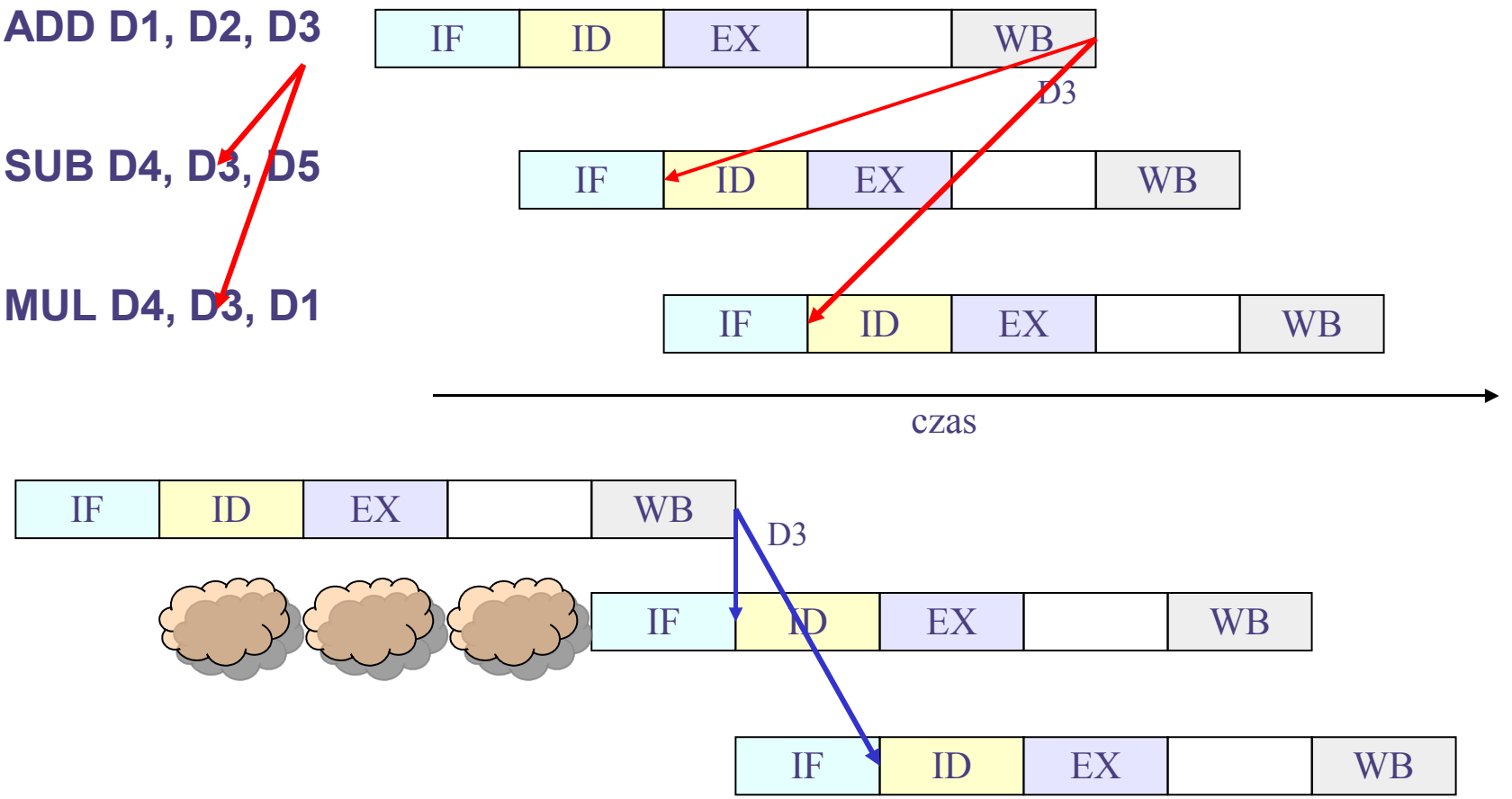
Structural Hazard Solution

- Hardware duplication
  - e.g. independent program (IF) and data (MEM) memory block (Harvard Architecture)
- Pipeline stall
  - simplest and some cases necessary, but always brings performance drop



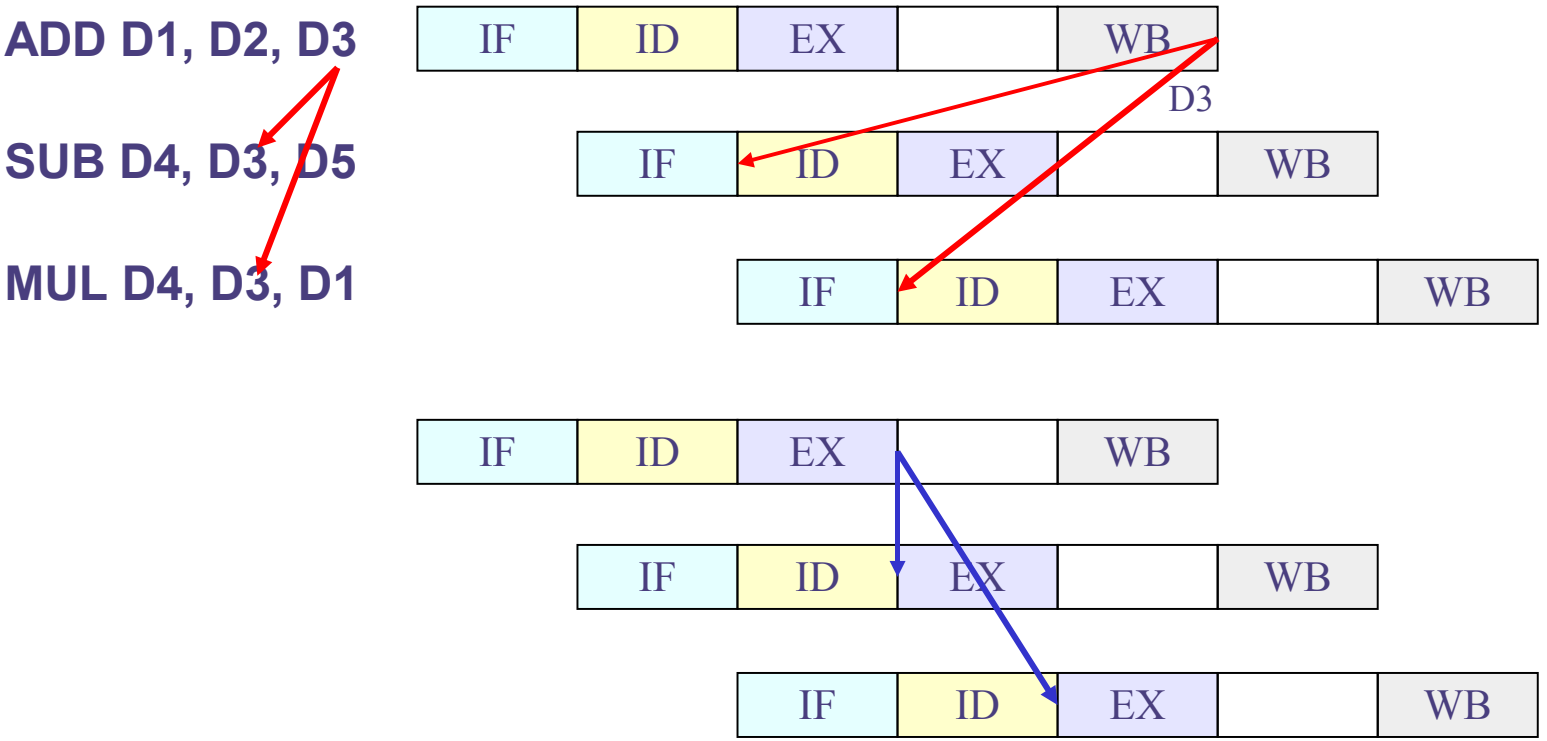
Data Hazard

- Attempt to use register content violating the register modification order
- Data hazards are very common – stall is wrong solution



Data Hazard Solution

- Forwarding
  - using up-to-date results directly from earlier execution stages, without waiting for final register update



Code Optimization

- Data hazards can be effectively eliminated by code optimization (by *optimizing compilers*)
- Architecture dependent or independent optimization

e.g. swapping two element of a table:

X[k] i X[k+1]

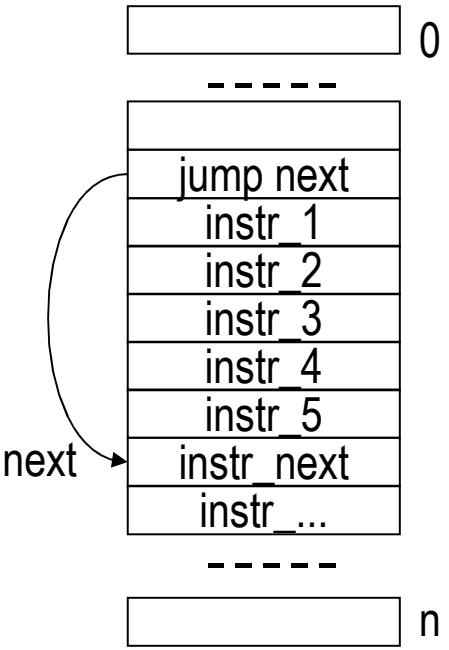
Register R3 contains addres of X[k] table element

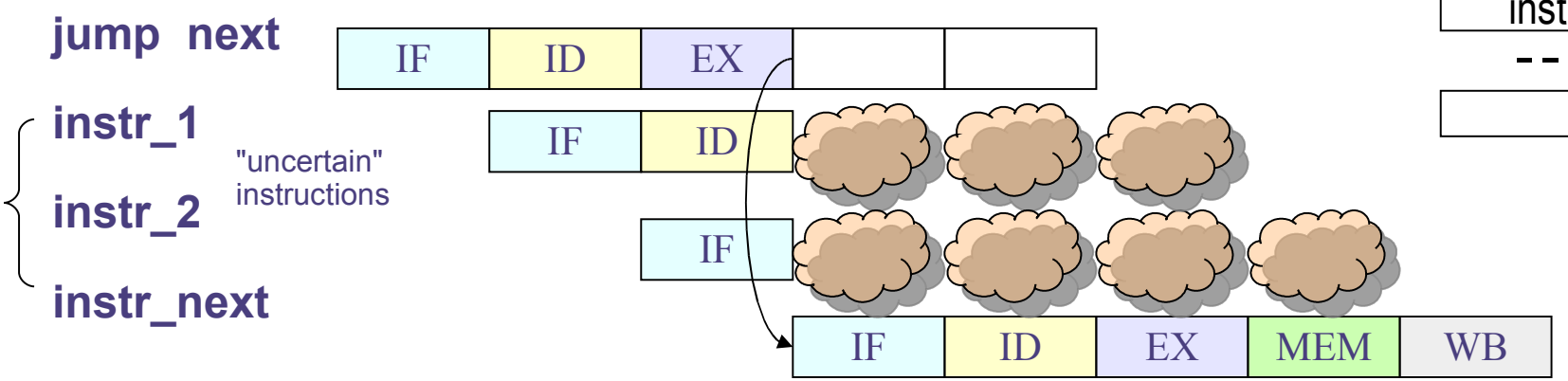
```
Before:                                  After:

lw R1,0(R3)   * R1     = X[k]     lw R1,0(R3)   * R1     = X[k]
lw R2,4(R3)   * R2     = X[k+1]   lw R2,4(R3)   * R2     = X[k+1]
sw R2,0(R3)   * X[k]   = R2       sw R1,4(R3)   * X[k+1]= R1
sw R1,4(R3)   * X[k+1]= R1        sw R2,0(R3)   * X[k]   = R2
```

Control Hazard

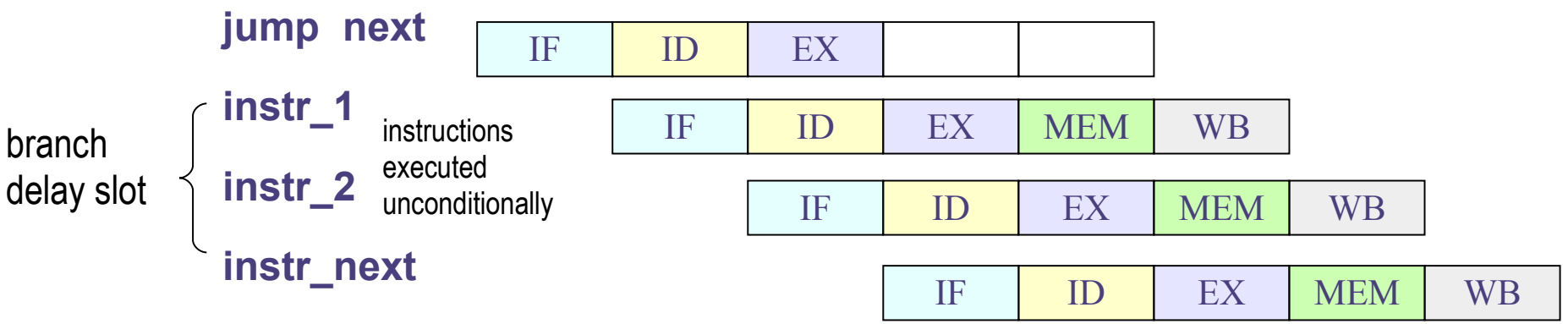
- Every branch (or jump) breaks instruction sequence
- Conditional branch (or jump) delay the moment of next instruction execution until the condition is evaluated
- Pipeline processor may allow some "uncertain" instructions to be fetched and processed and invalidated if necessary





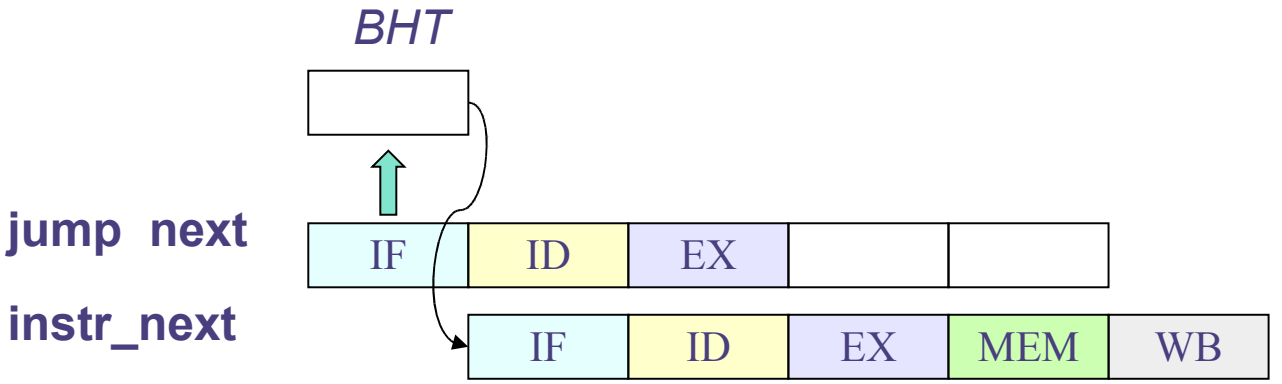
Delayed Branch

- Instructions between the branch and the target, that entered the pipeline (in delay slot), will be executed unconditionally
- Instruction in delay branch slot must be chosen carefully (useful or at least harmless - NOP)



Branch History Table

- BHT contains addresses of recently executed branches and their last target – during IF stage the branch and the target can be identified just in time
- Conditional branch validates the entry in BHT – the target instruction may be invalidated if necessary
- Unsuccessful predictions are limited to starts and ends of program loops

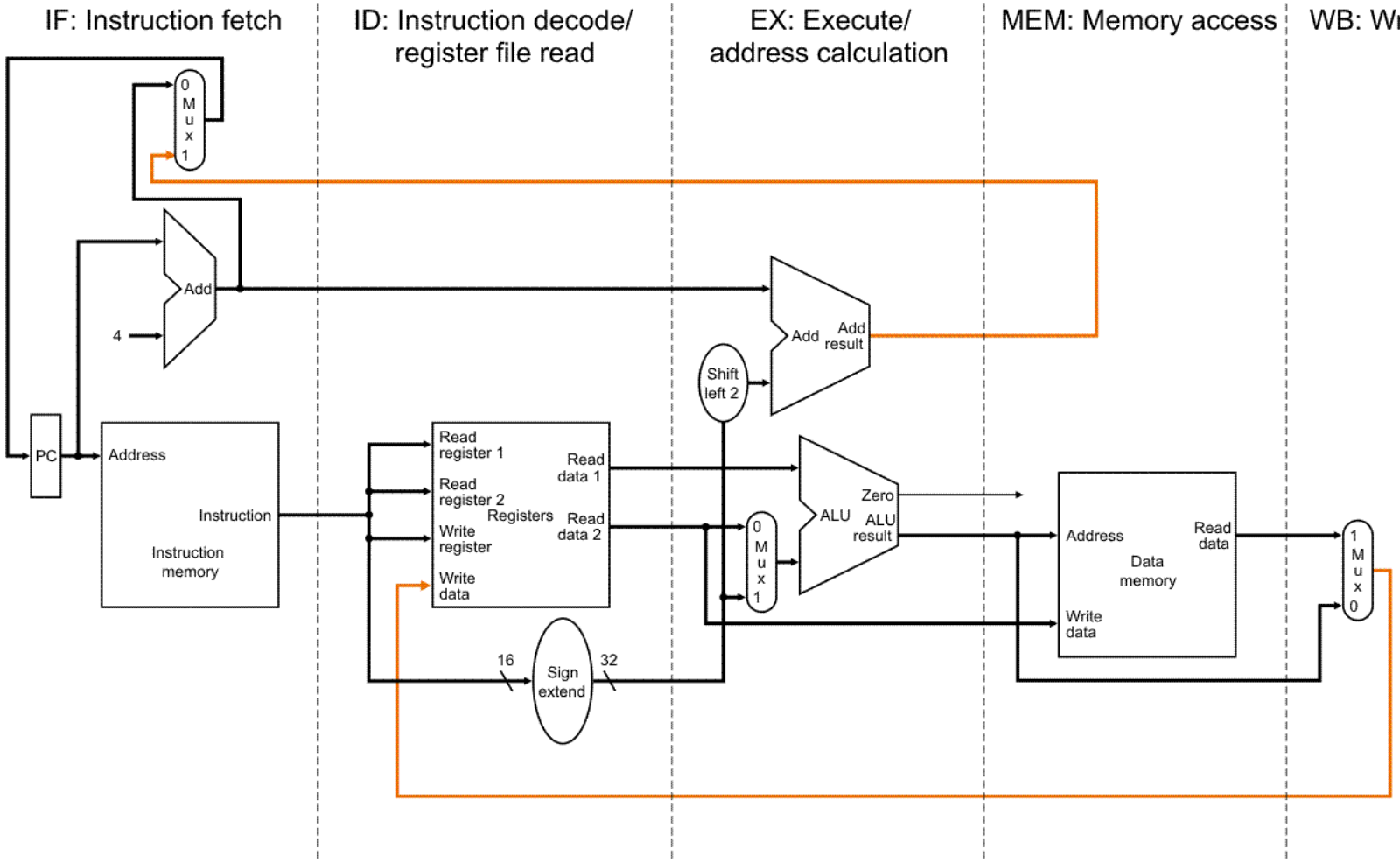


Pipelined Processing Summary

- Idea: 1 instruction / clock cycle (CPI)
- Instruction machine cycle is the same (5 stages long), but effective CPI is 1
- Hazards (resource, data, control) lower the pipeline performance: CPI>1
- Several instructions are always processed at a time
- All hardware resources are used at a time (almost)
- Software optimization is crucial to pipelined processing

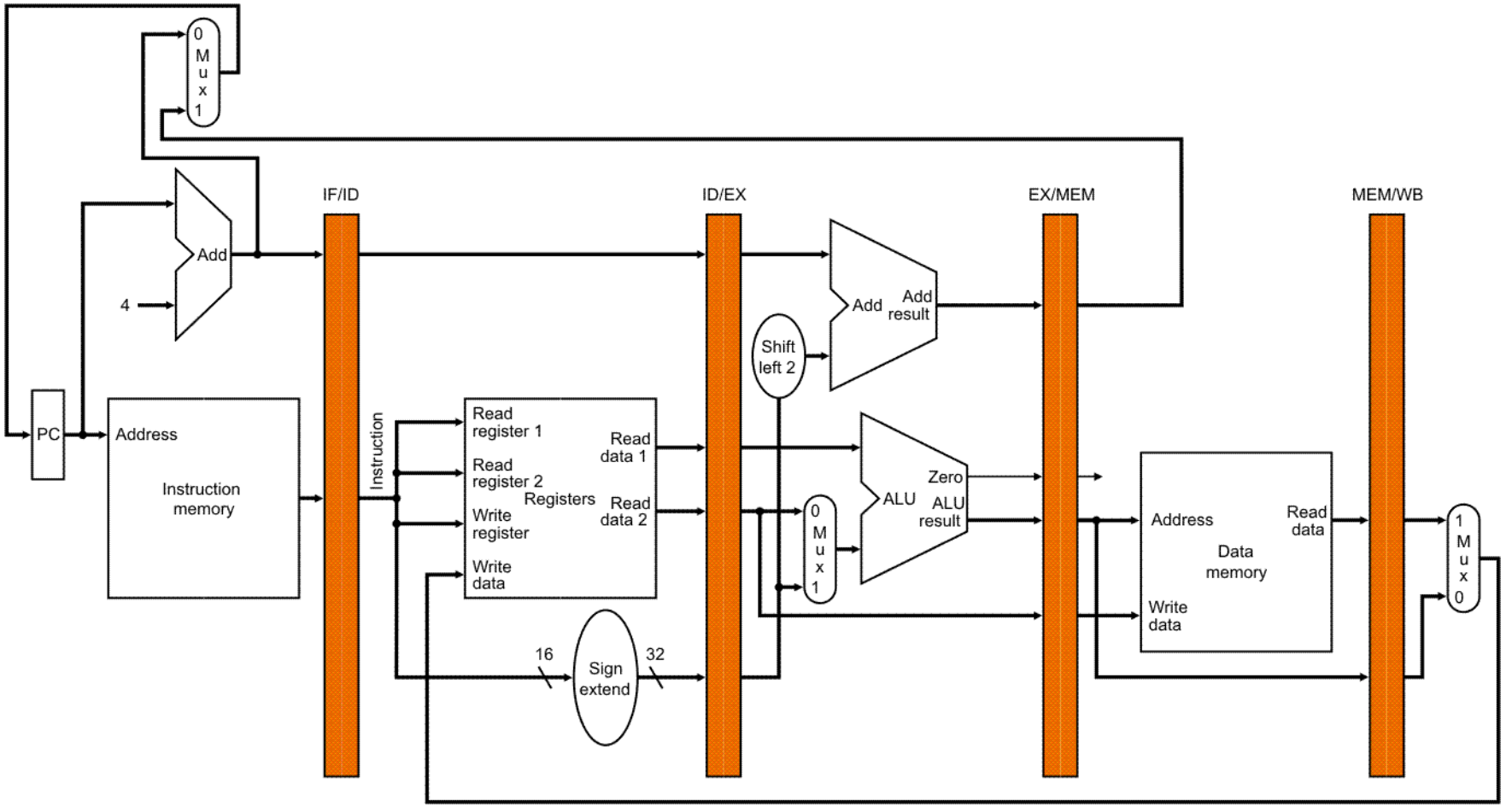
Starting point: *Single-Cycle Arch.*

- Data flow can be clearly identified (but asynchronous)
- Independent stages and independent hardware



Pipelined Architecture

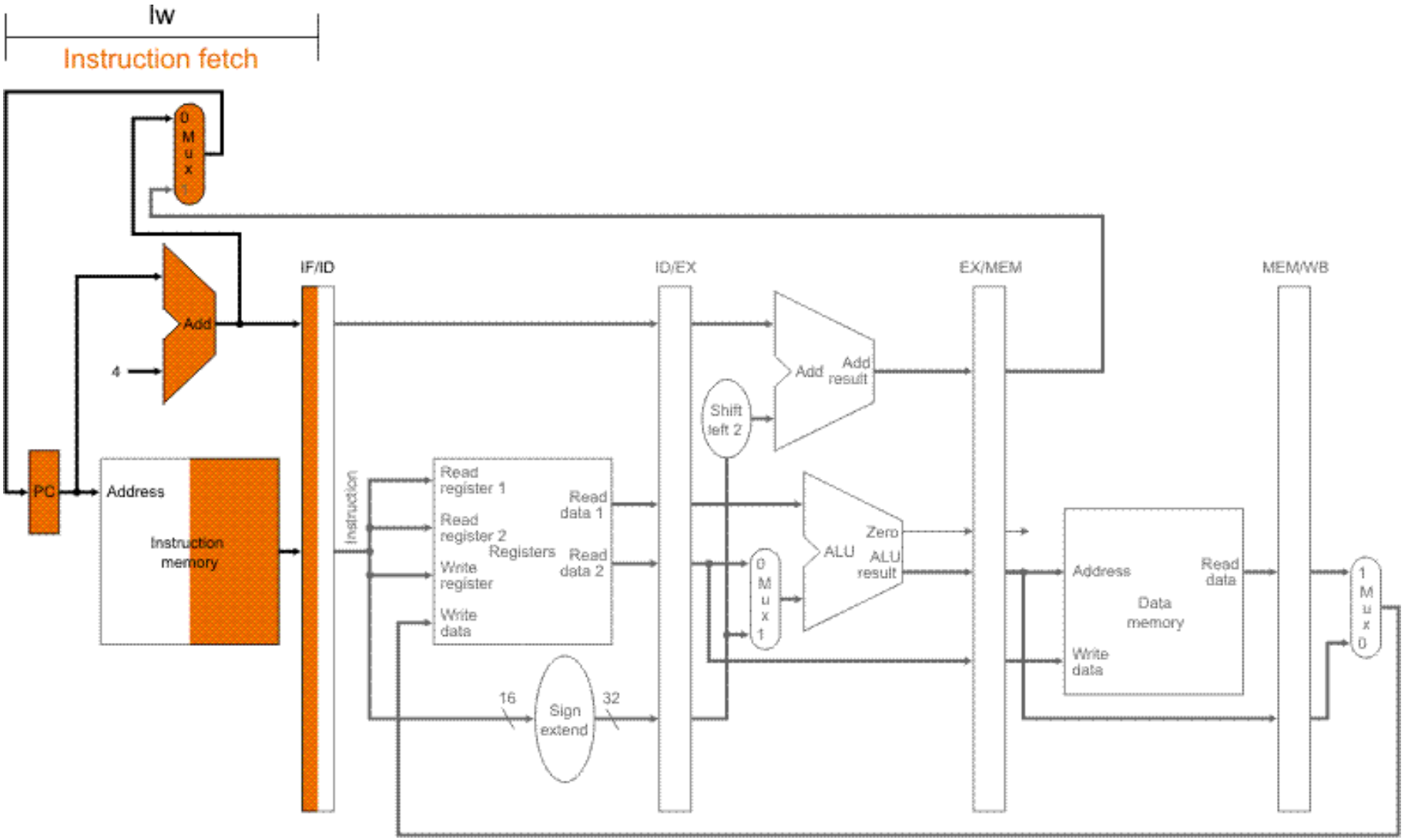
- Intermediate registers to synchronize the data flow



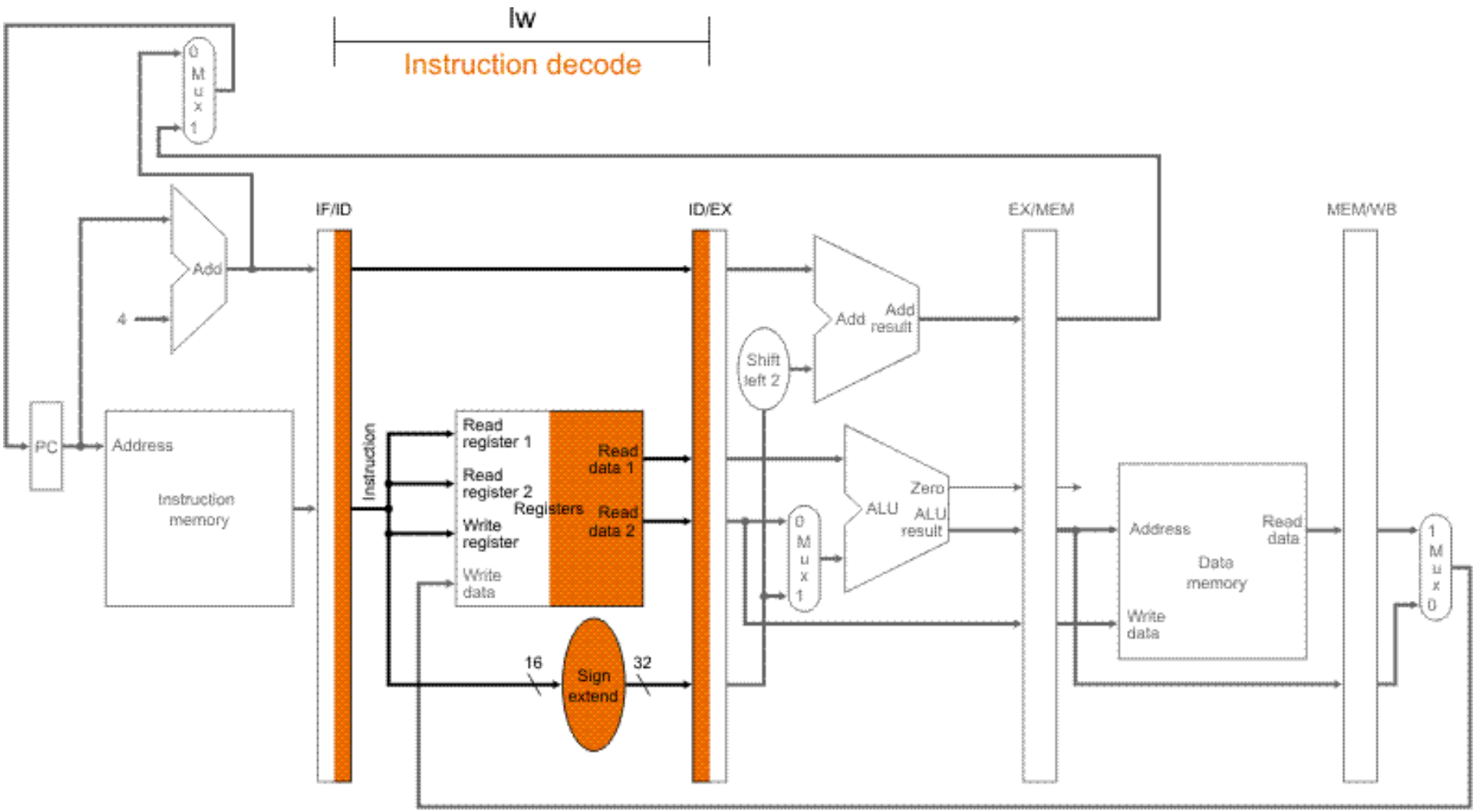
Pipelined Hardware Implications

- Hardware complexity – resource duplication (adders, memory) to avoid structural hazards
- Introduction of "long" intermediate registers to synchronize data flow between stages of instruction execution
- Instructions can use any resources only one time, more complex tasks (with reuse of resources) are impossible
- Instruction list and functionality is reduced (RISC) – simple addressing modes - Load/Store architecture
- Performance increase is achieved at the cost of having simpler instructions – programs (machine code) must be longer
- Control unit implementation is simple – a consequence of reduced instruction set

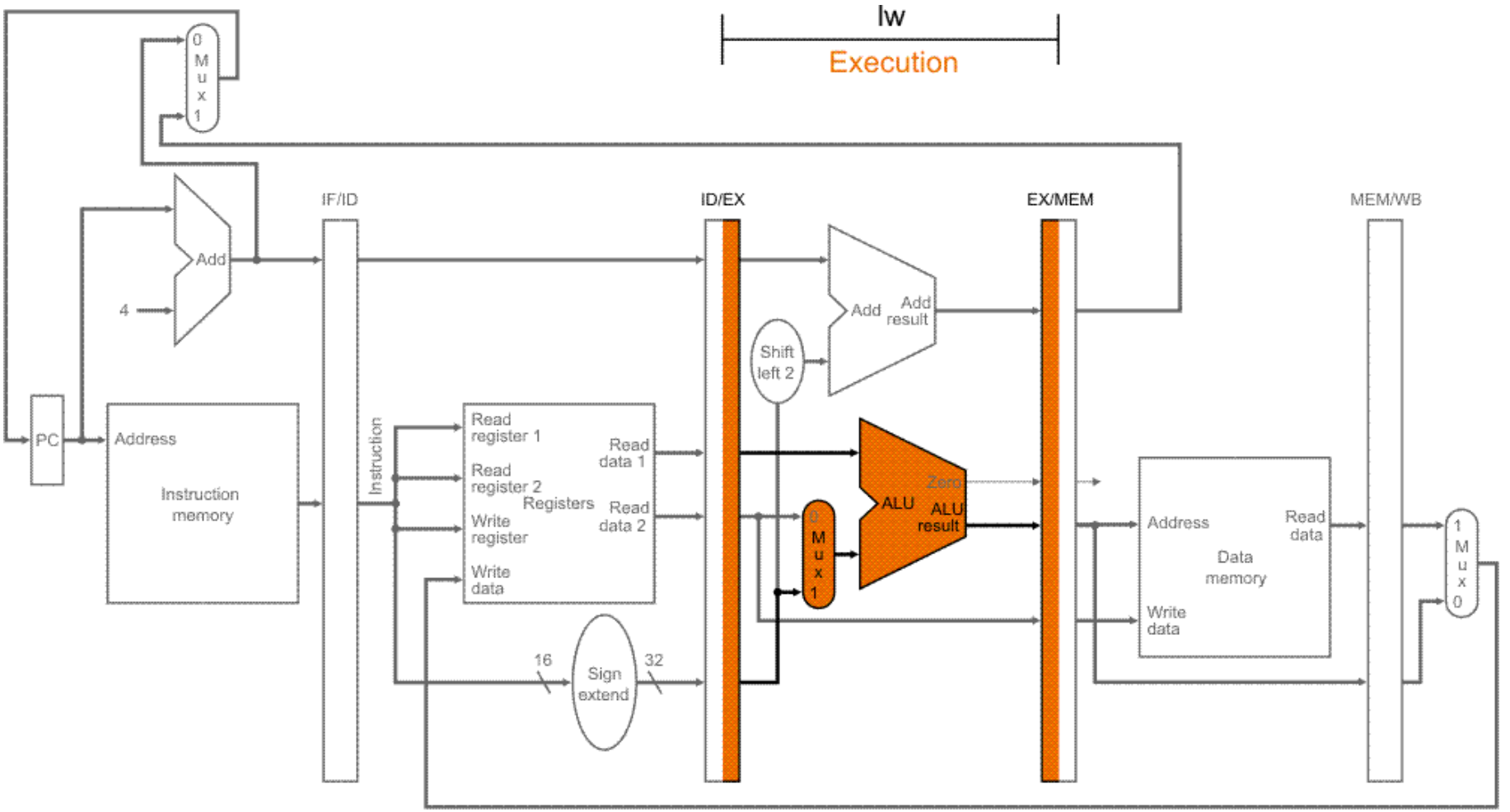
Load Word (LW) – demonstration (1)



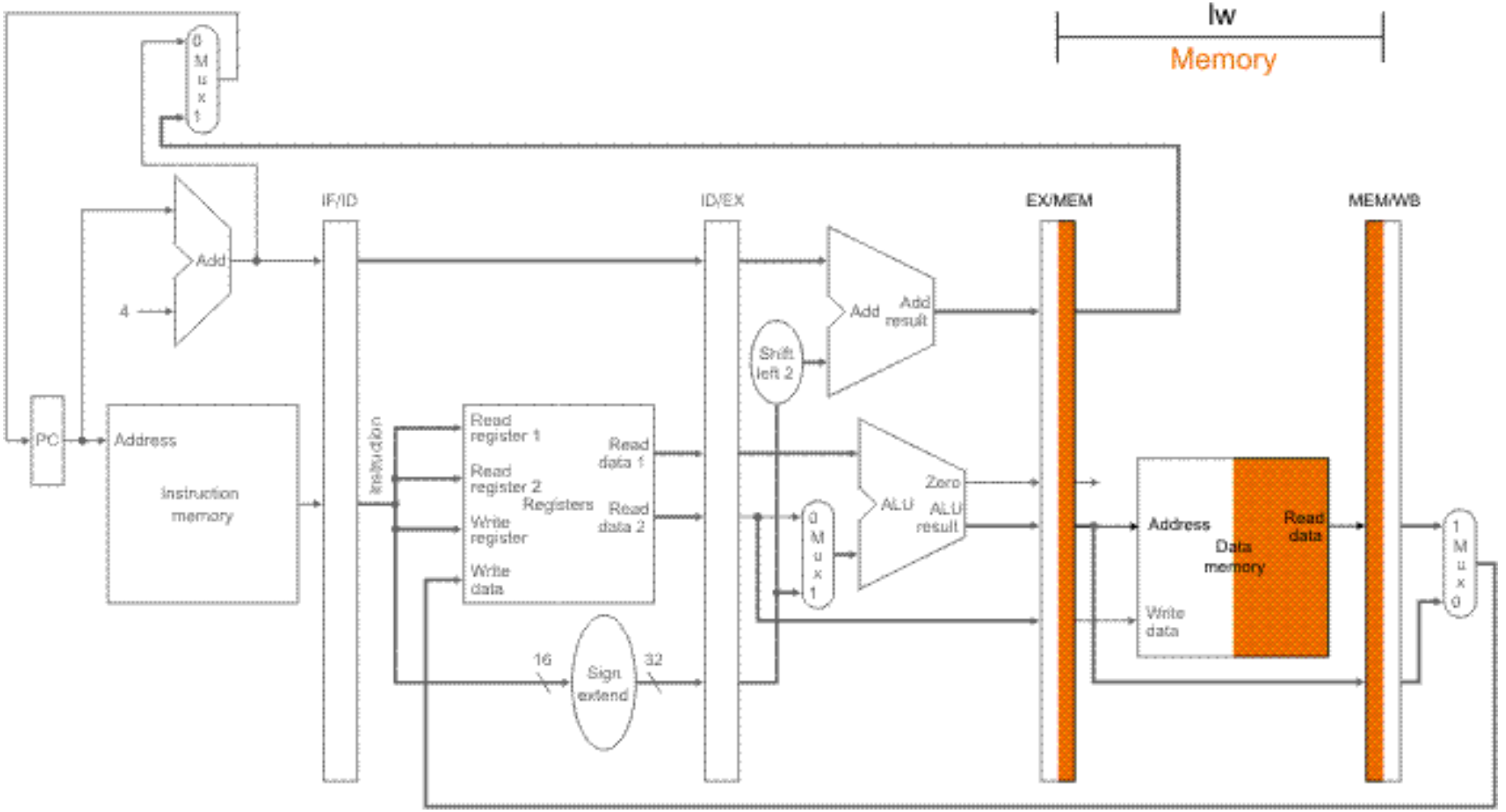
LW (2)


lw
Instruction decode
IF/ID
ID/EX
EX/MEM
MEM/WB
M u x
Add
4
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
16
Sign extend
32
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Data memory
Read data
Write data

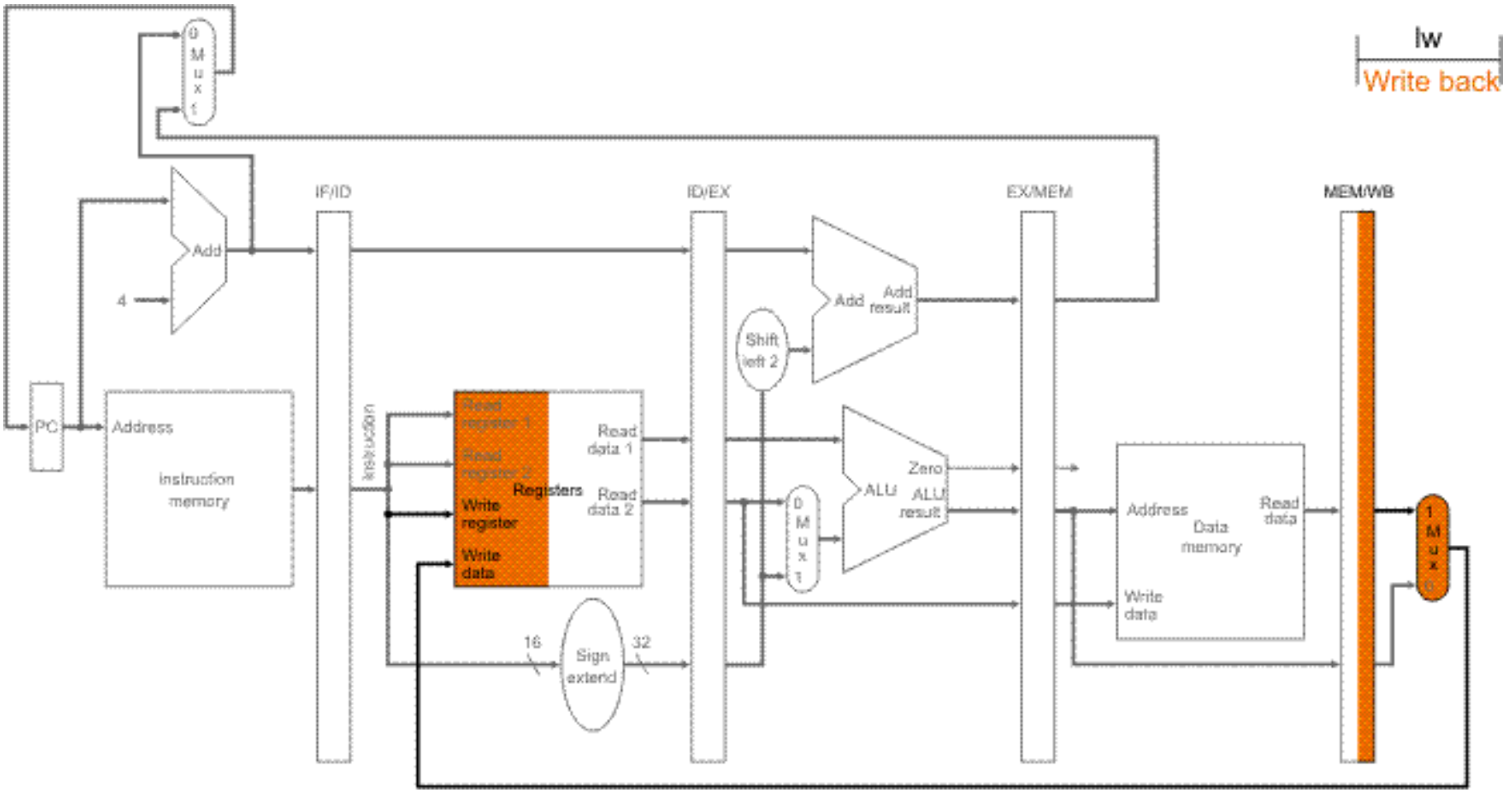
LW (3)


lw
Execution
IF/ID
ID/EX
EX/MEM
MEM/WB
0
M
u
x
1
Add
4
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
ALU
Zero
ALU result
16
Sign extend
32
Address
Data memory
Read data
Write data
1
M
u
x
0

LW (4)

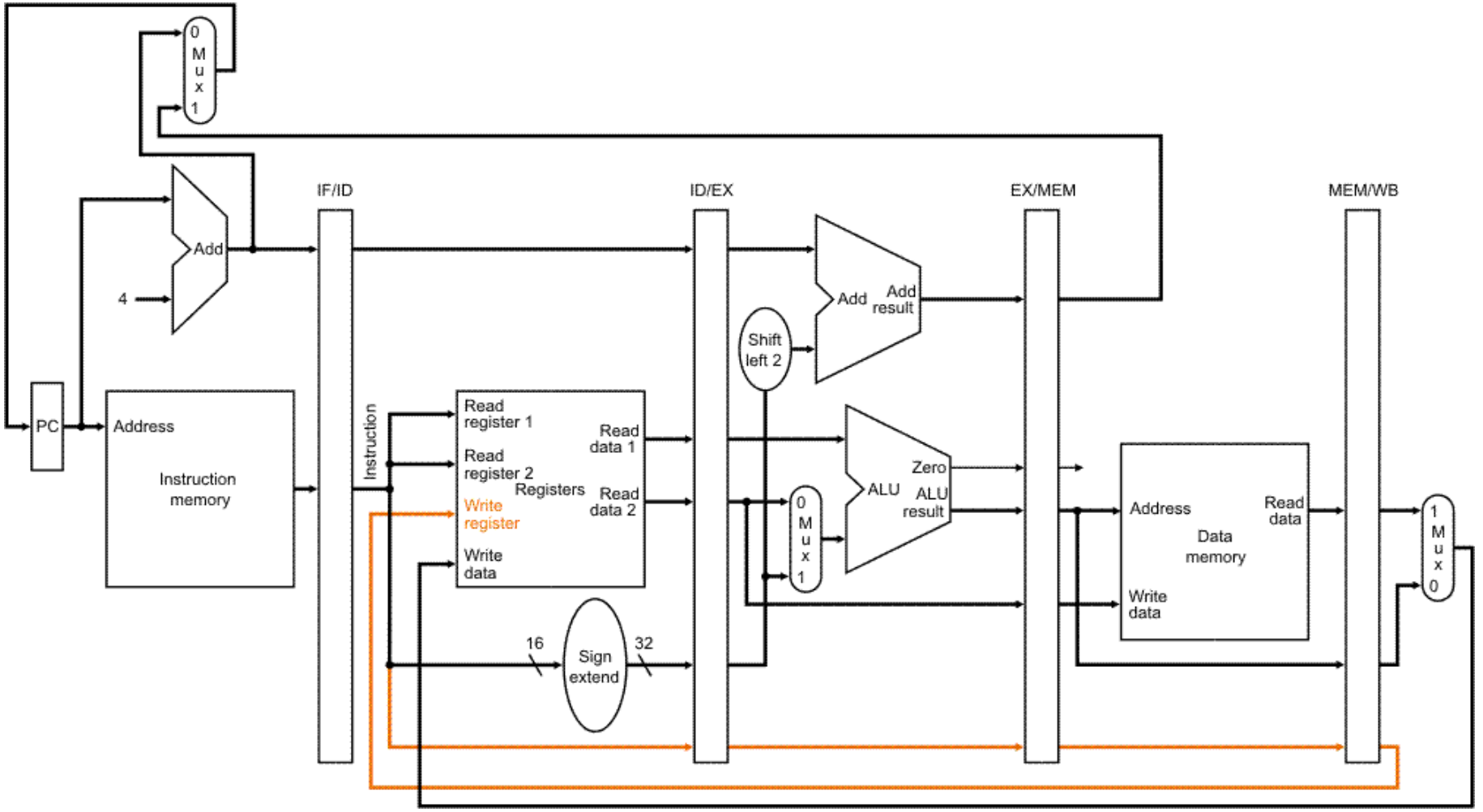

lw
Memory
IF/ID
ID/EX
EX/MEM
MEM/WB
M u x
Add
4
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
16
Sign extend
32
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Data memory
Read data
Write data

LW (5)


lw
Write back
IF/ID
ID/EX
EX/MEM
MEM/WB
M u x
Add
4
PC
Address
Instruction memory
Instruction
Registers
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
16
Sign extend
32
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Data memory
Read data
Write data

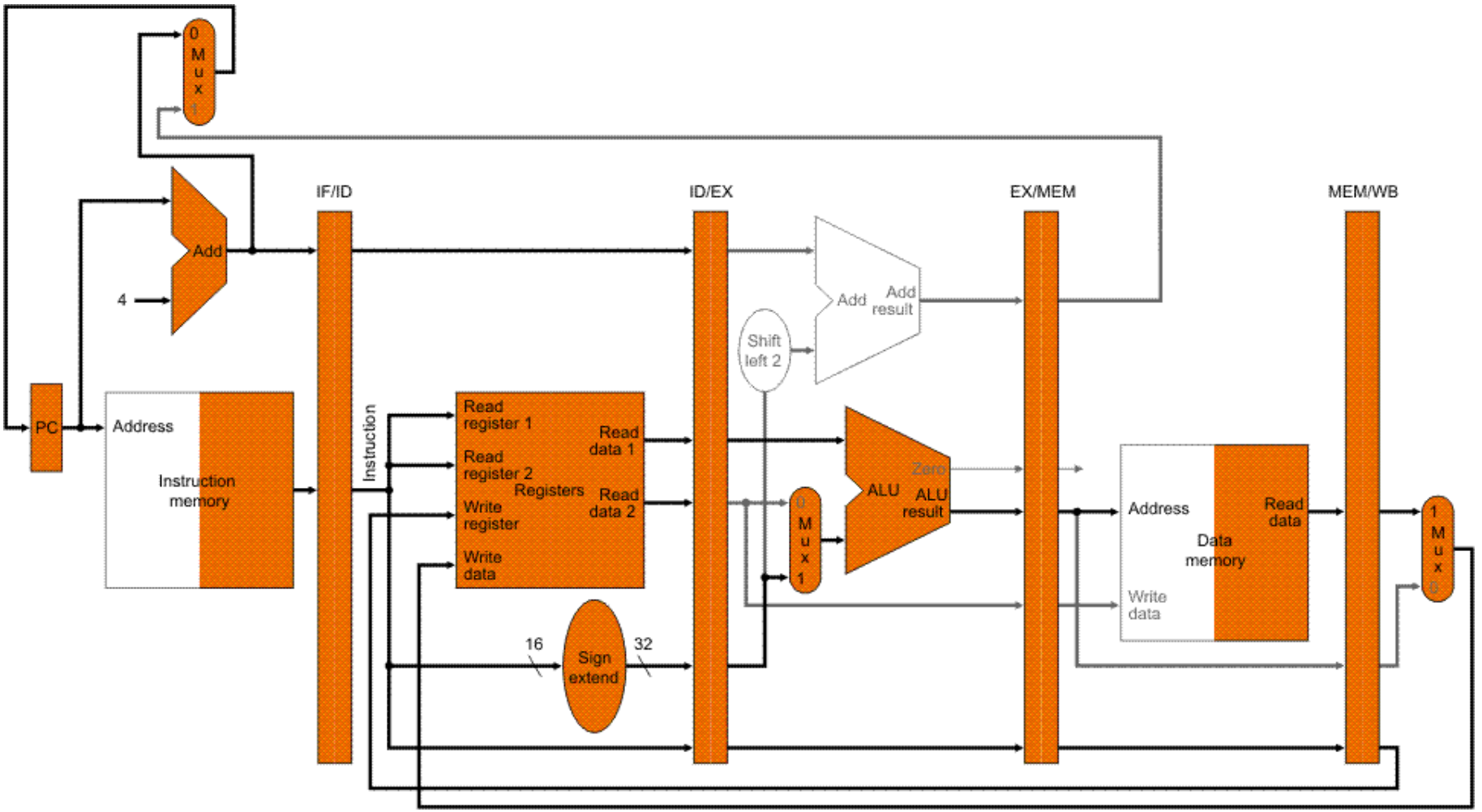
Register Update Correction (LW, R-type)

- Instruction must "carry" the register number to be modified through the whole pipeline

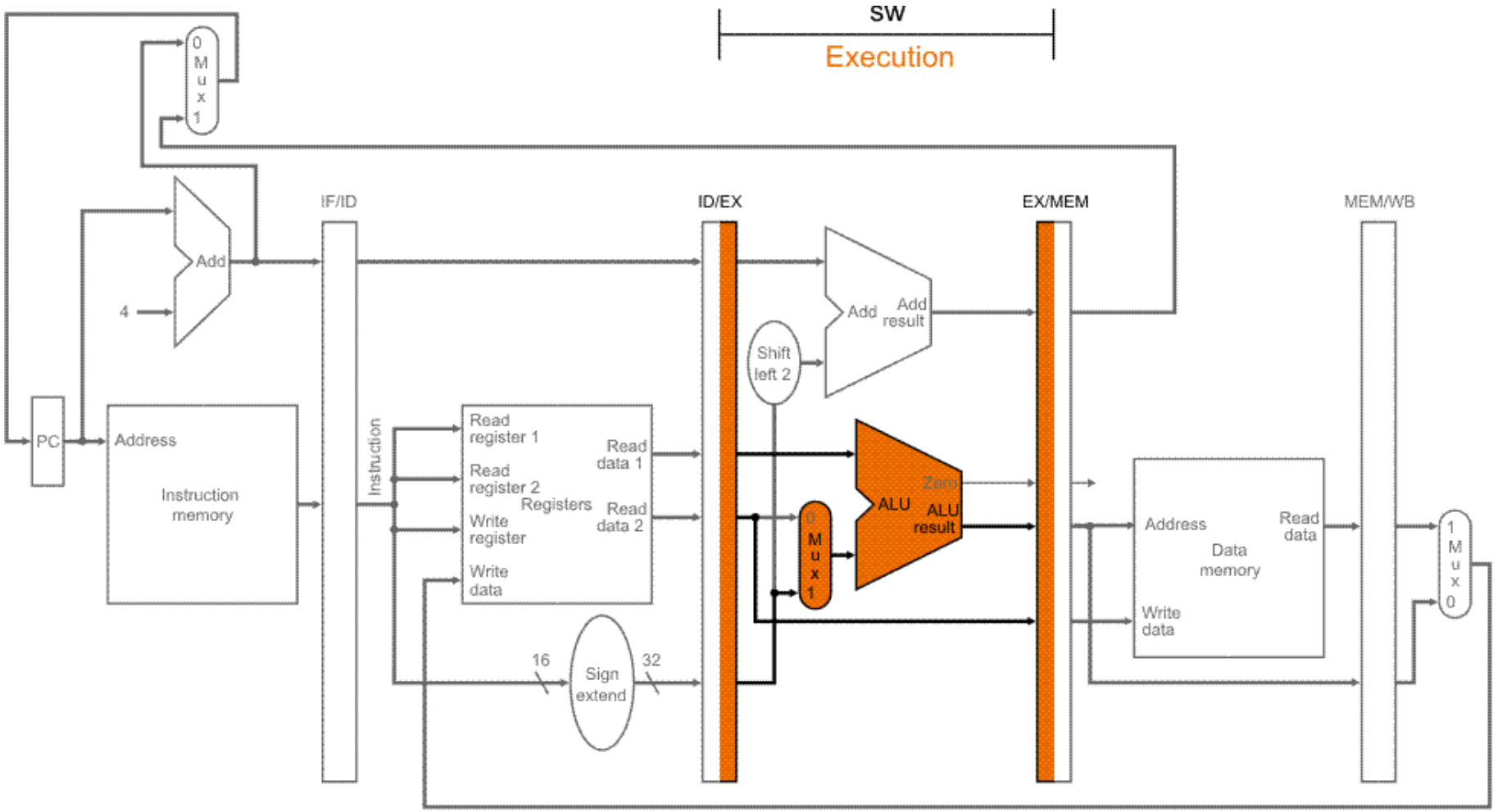


LW – Resource Utilization Example

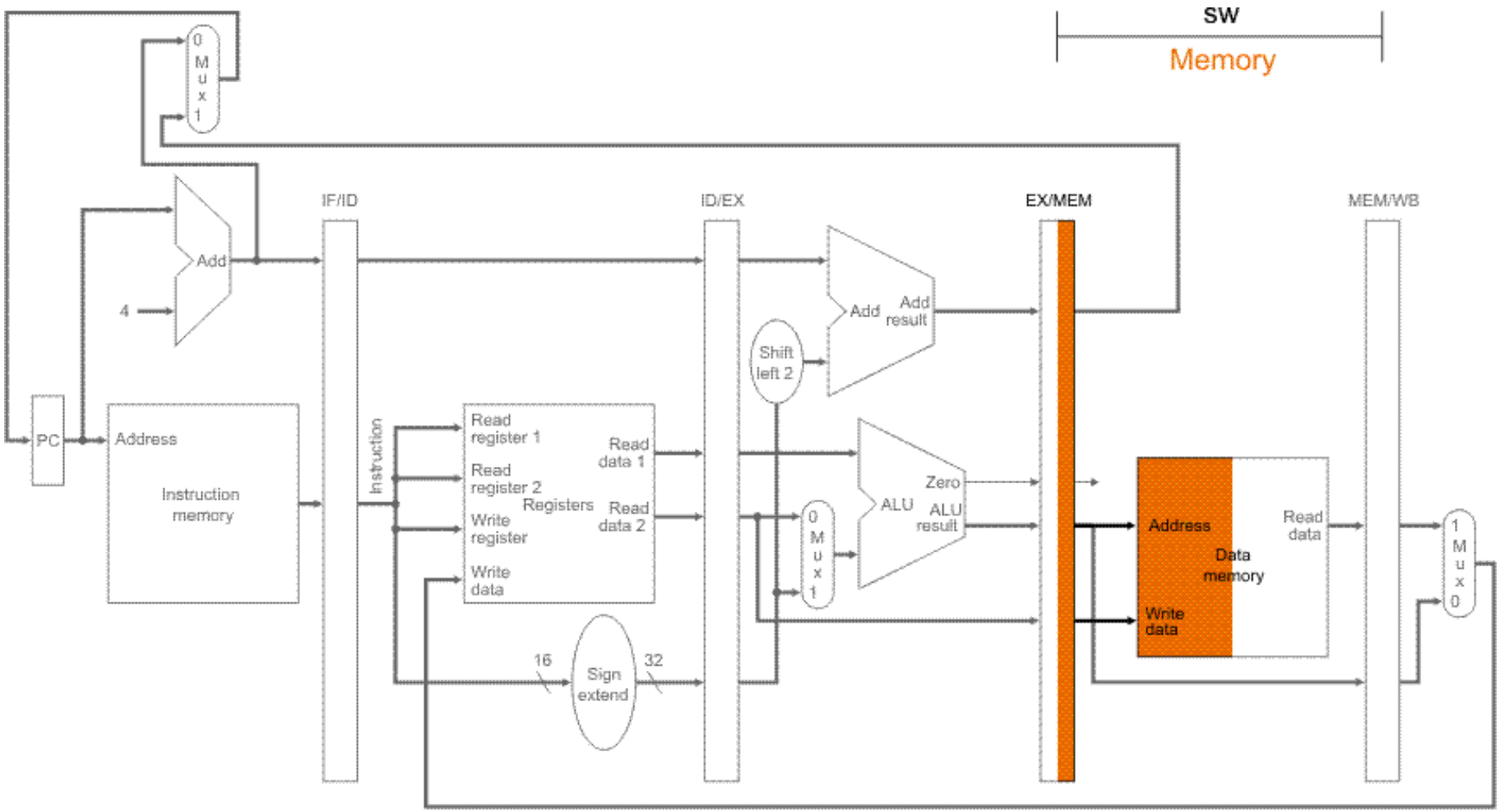
- Resource utilization in all 5 stages of execution



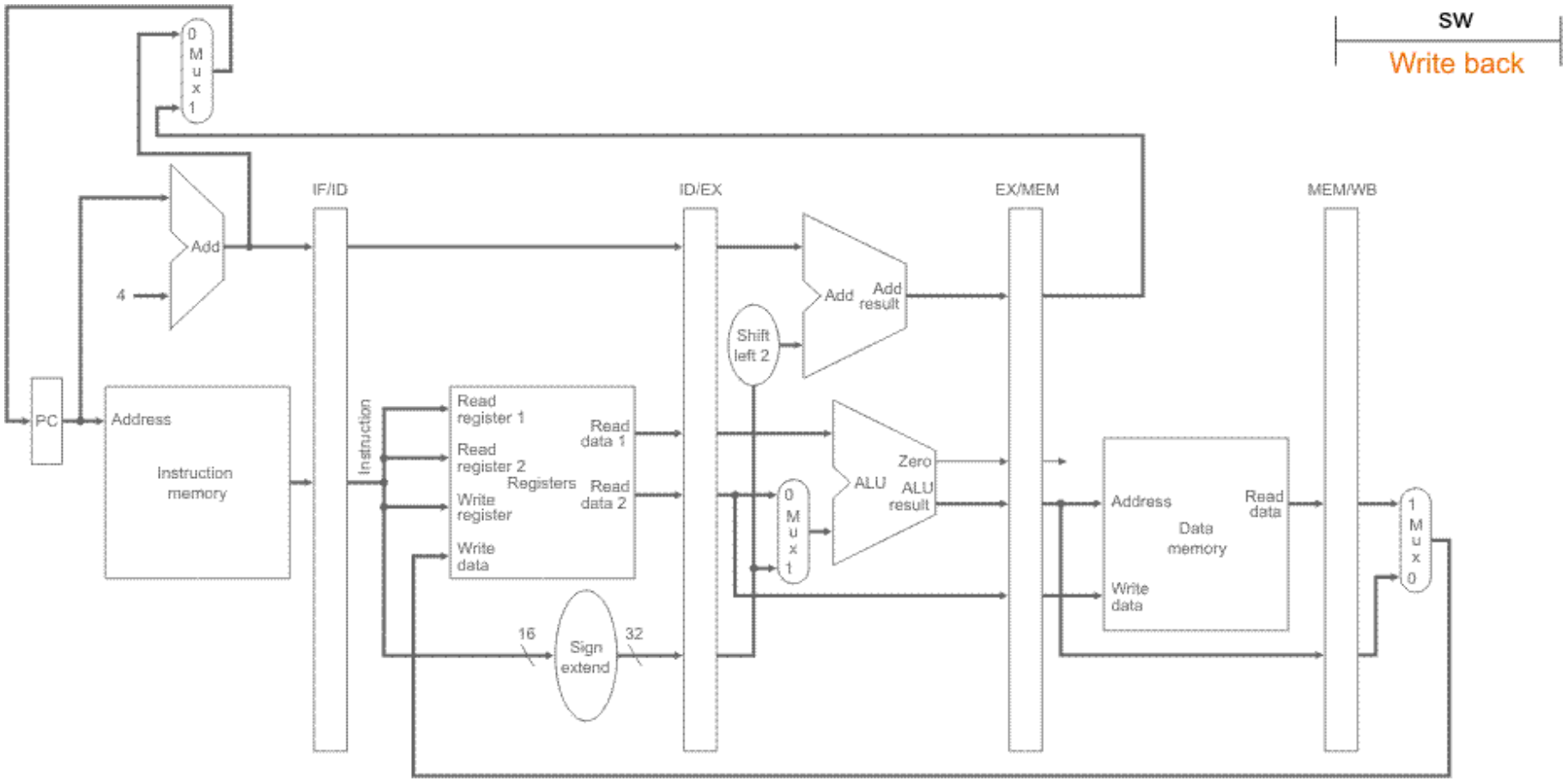
SW (3)


SW
Execution
IF/ID
ID/EX
EX/MEM
MEM/WB
M u x
0
1
Add
4
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
16
Sign extend
32
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Data memory
Read data
Write data

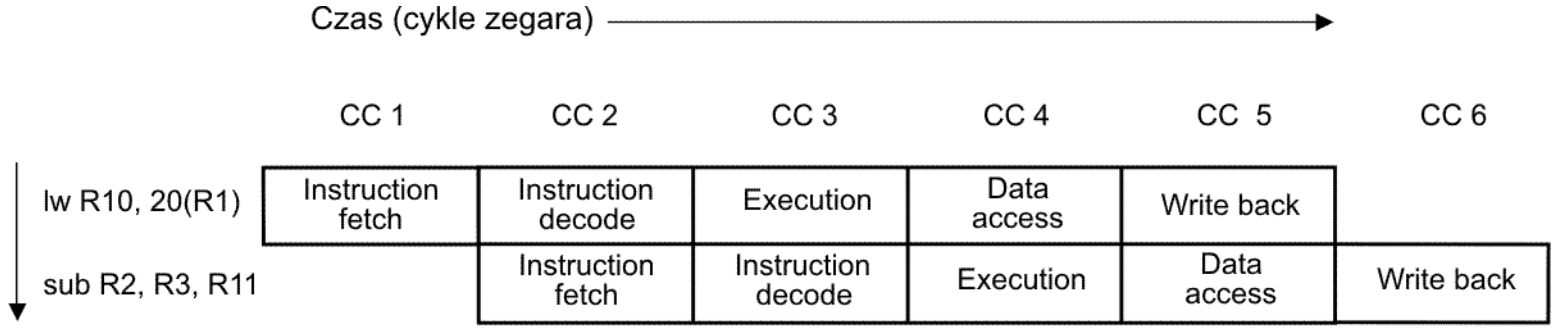
SW (4)

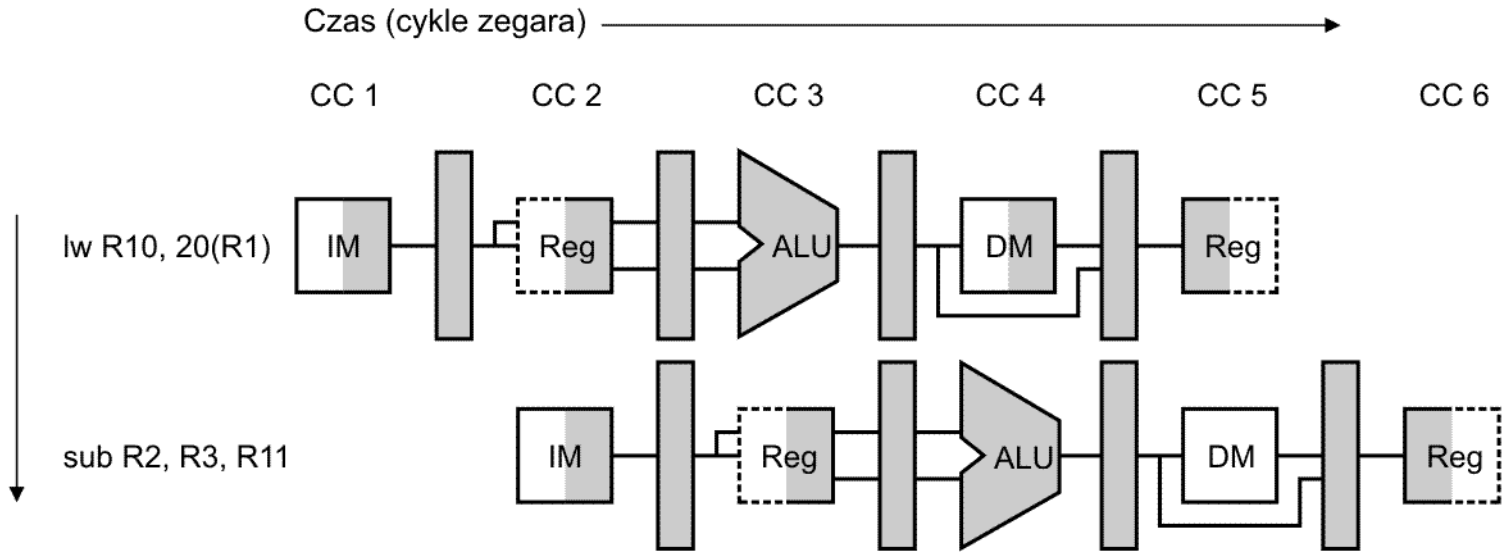

SW
Memory
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M u x
0
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
ALU
Zero
ALU result
16
Sign extend
32
Address
Data memory
Write data
Read data

SW (5) – „Empty cycle”

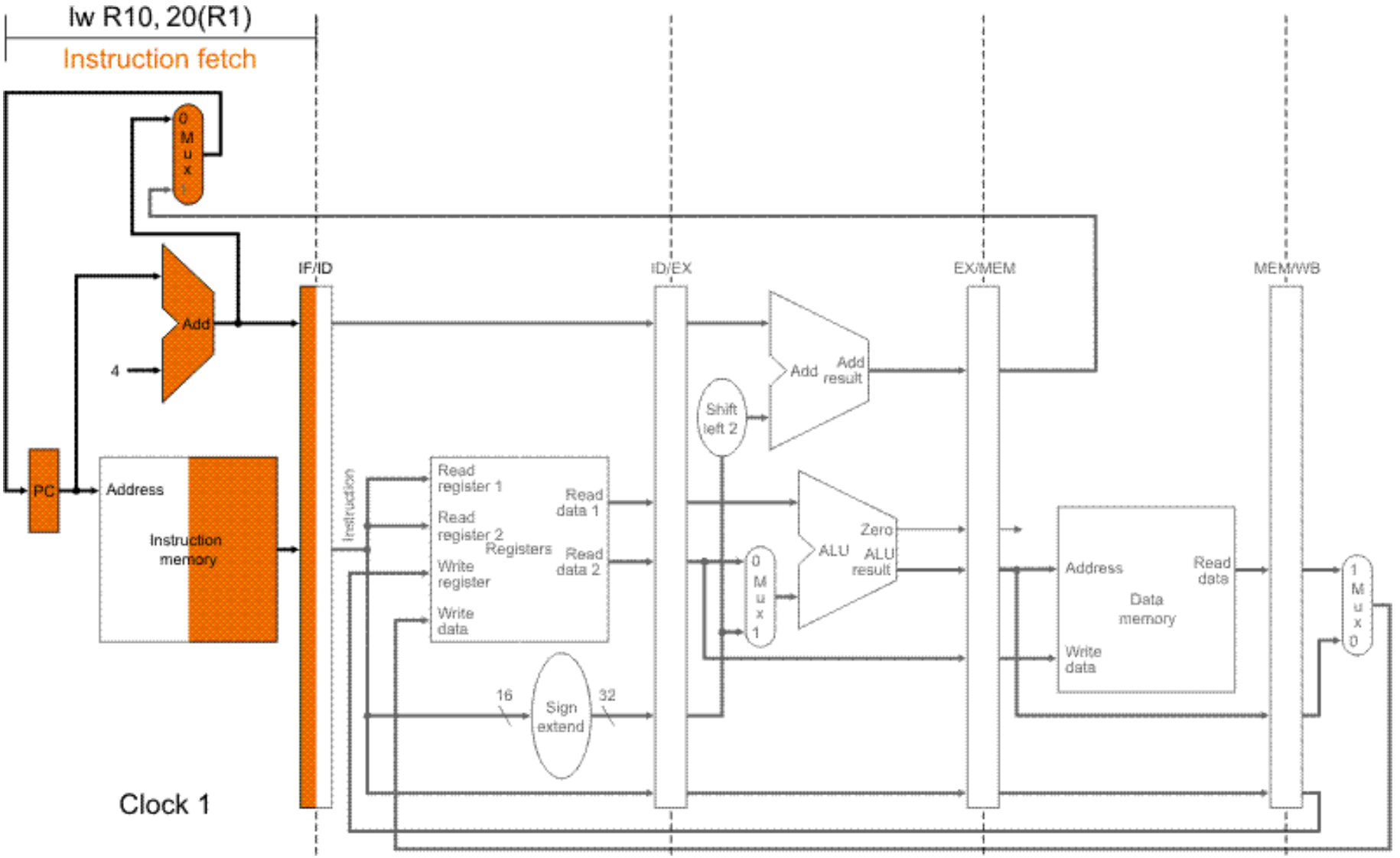


2 instructions – Demonstration

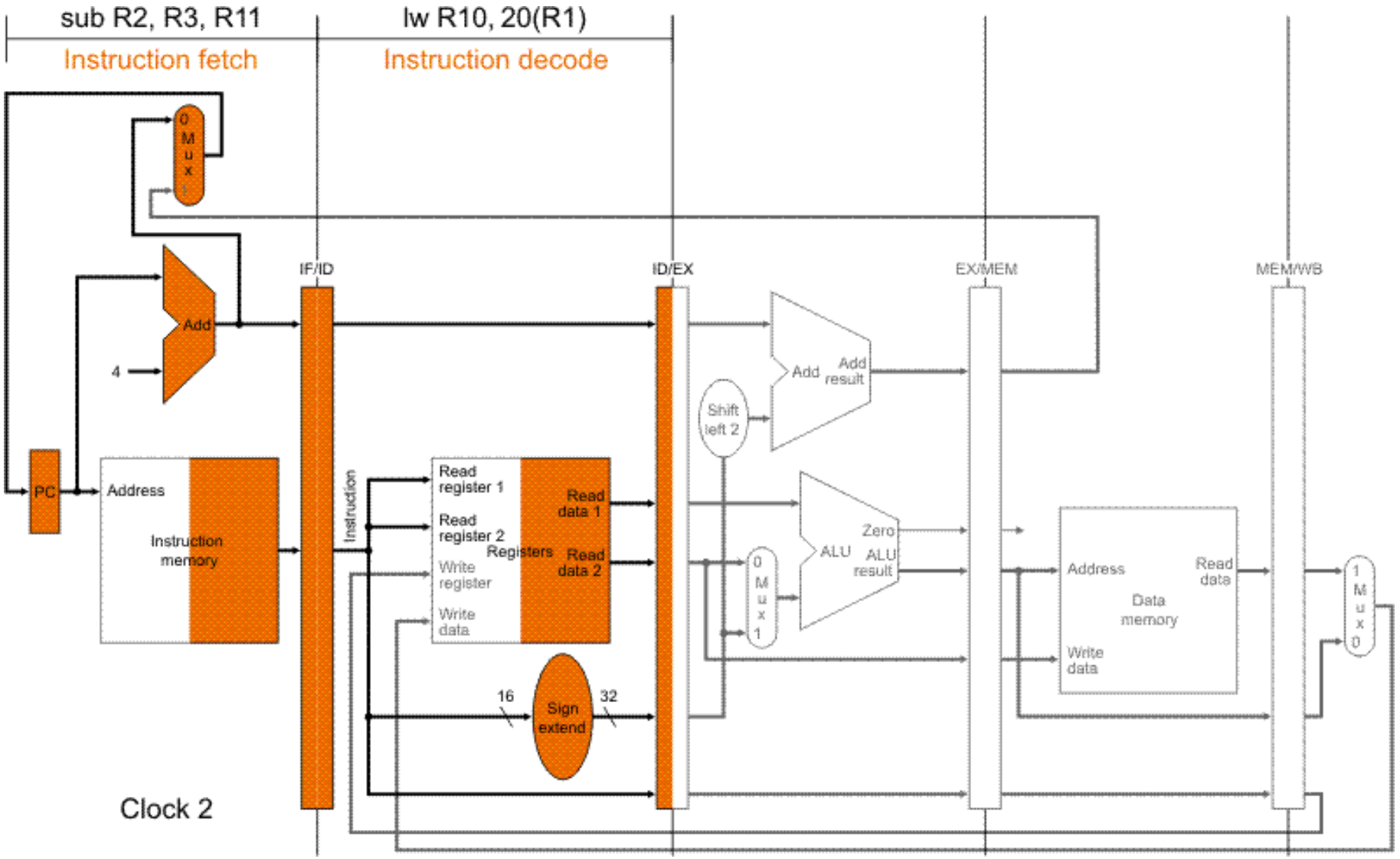




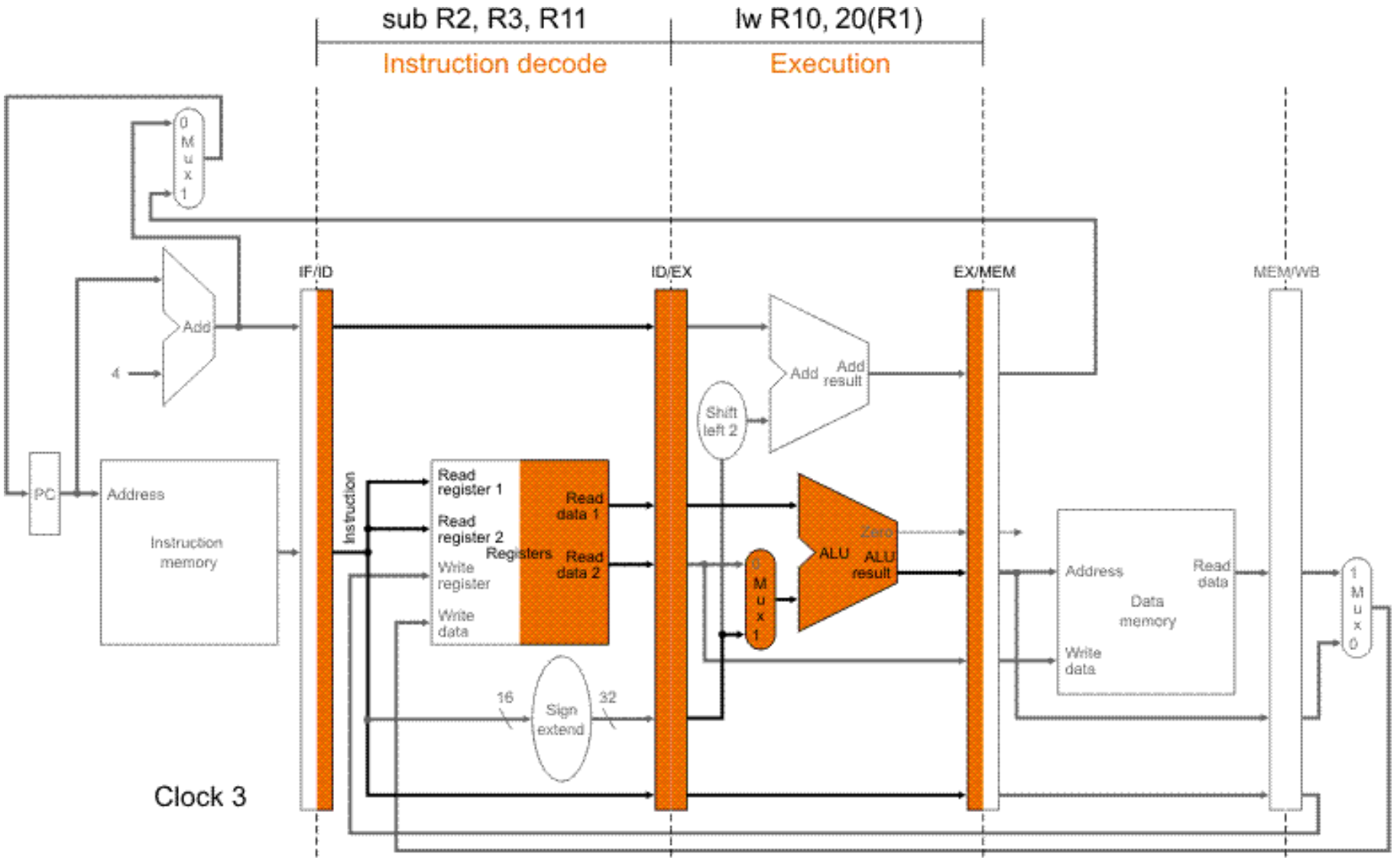
LW / SUB (1)


lw R10, 20(R1)
Instruction fetch
IF/ID
ID/EX
EX/MEM
MEM/WB
M u x
Add
4
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
ALU
Zero
ALU result
16
Sign extend
32
Address
Data memory
Read data
Write data
Clock 1

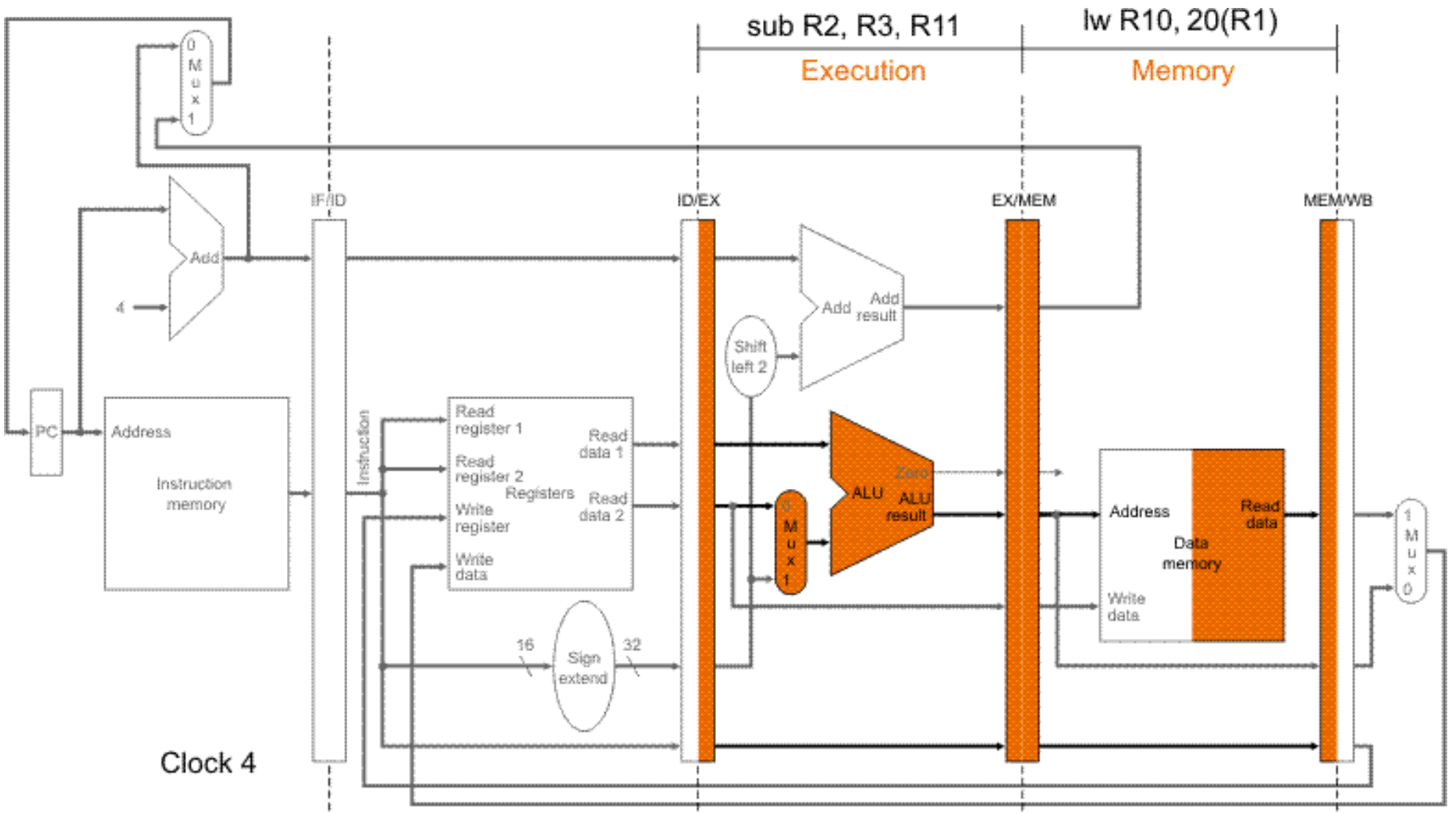
LW / SUB (2)


sub R2, R3, R11
lw R10, 20(R1)
Instruction fetch
Instruction decode
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
16
Sign extend
32
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Data memory
Read data
Write data
Clock 2

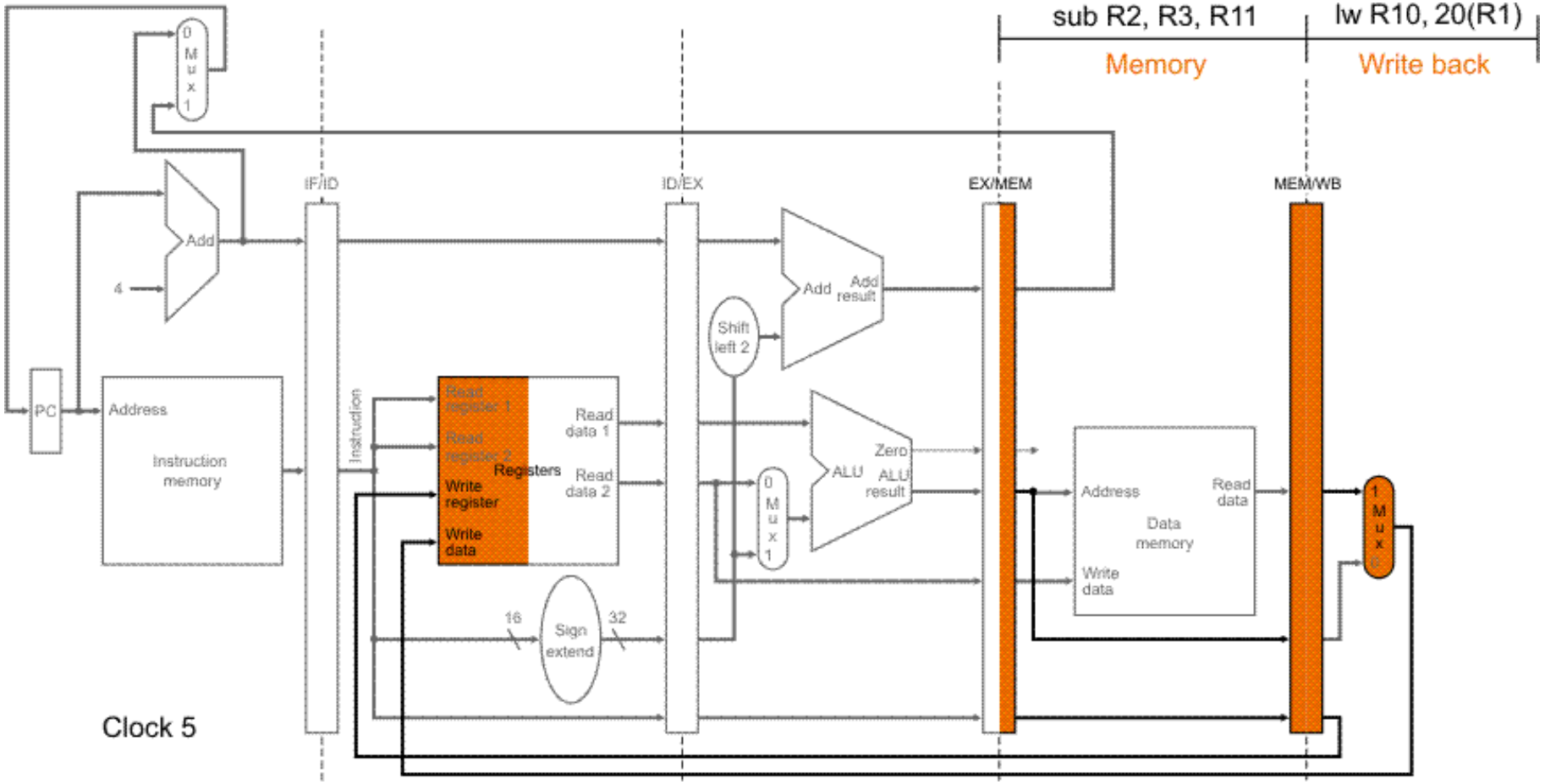
LW / SUB (3)


sub R2, R3, R11
lw R10, 20(R1)
Instruction decode
Execution
IF/ID
ID/EX
EX/MEM
MEM/WB
M u x
0
1
Add
4
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Data memory
Read data
Write data
16
Sign extend
32
Clock 3

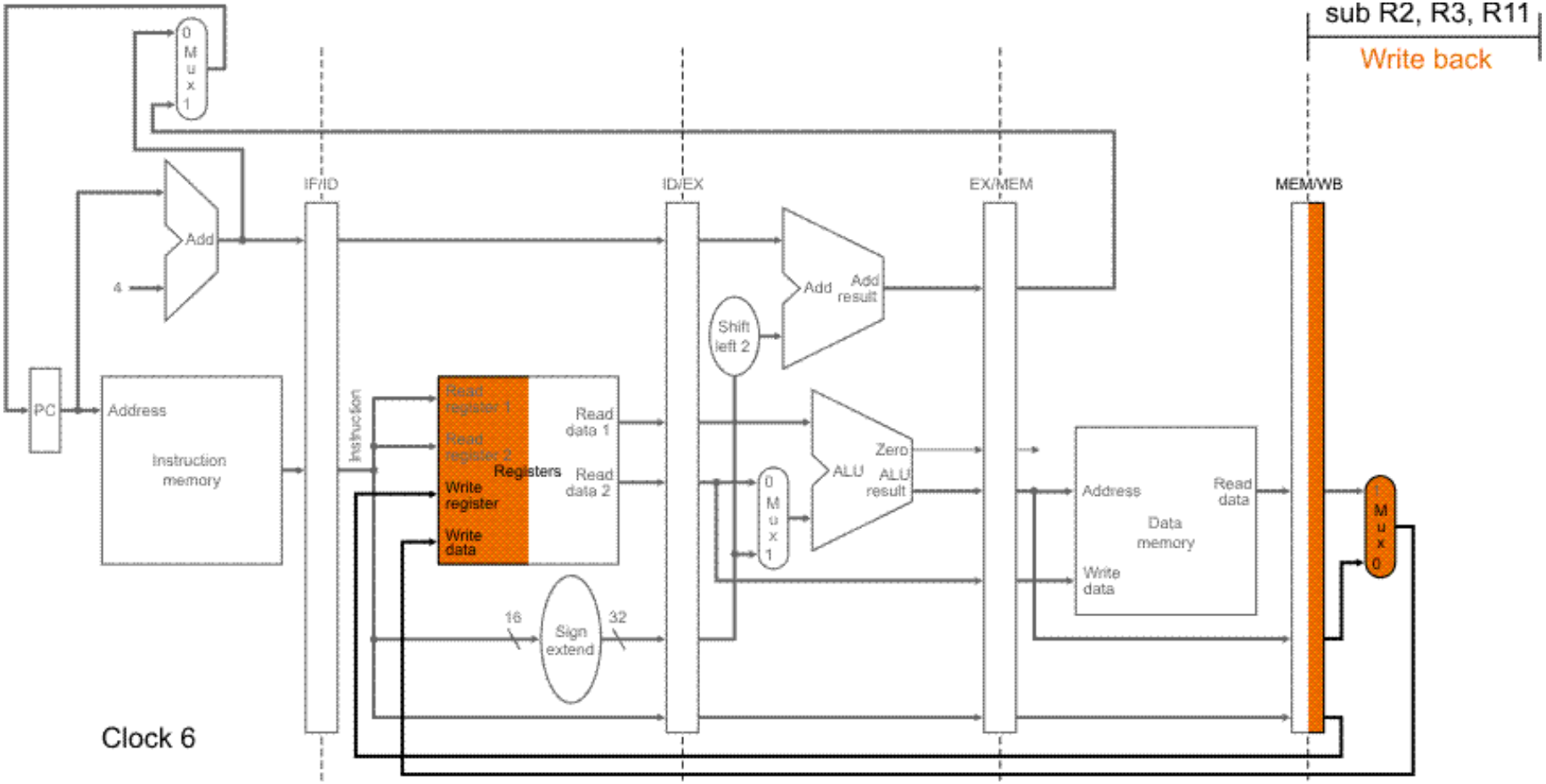
LW / SUB (4)


sub R2, R3, R11
lw R10, 20(R1)
Execution
Memory
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M u x
0
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Data memory
Read data
Write data
16
Sign extend
32
Clock 4

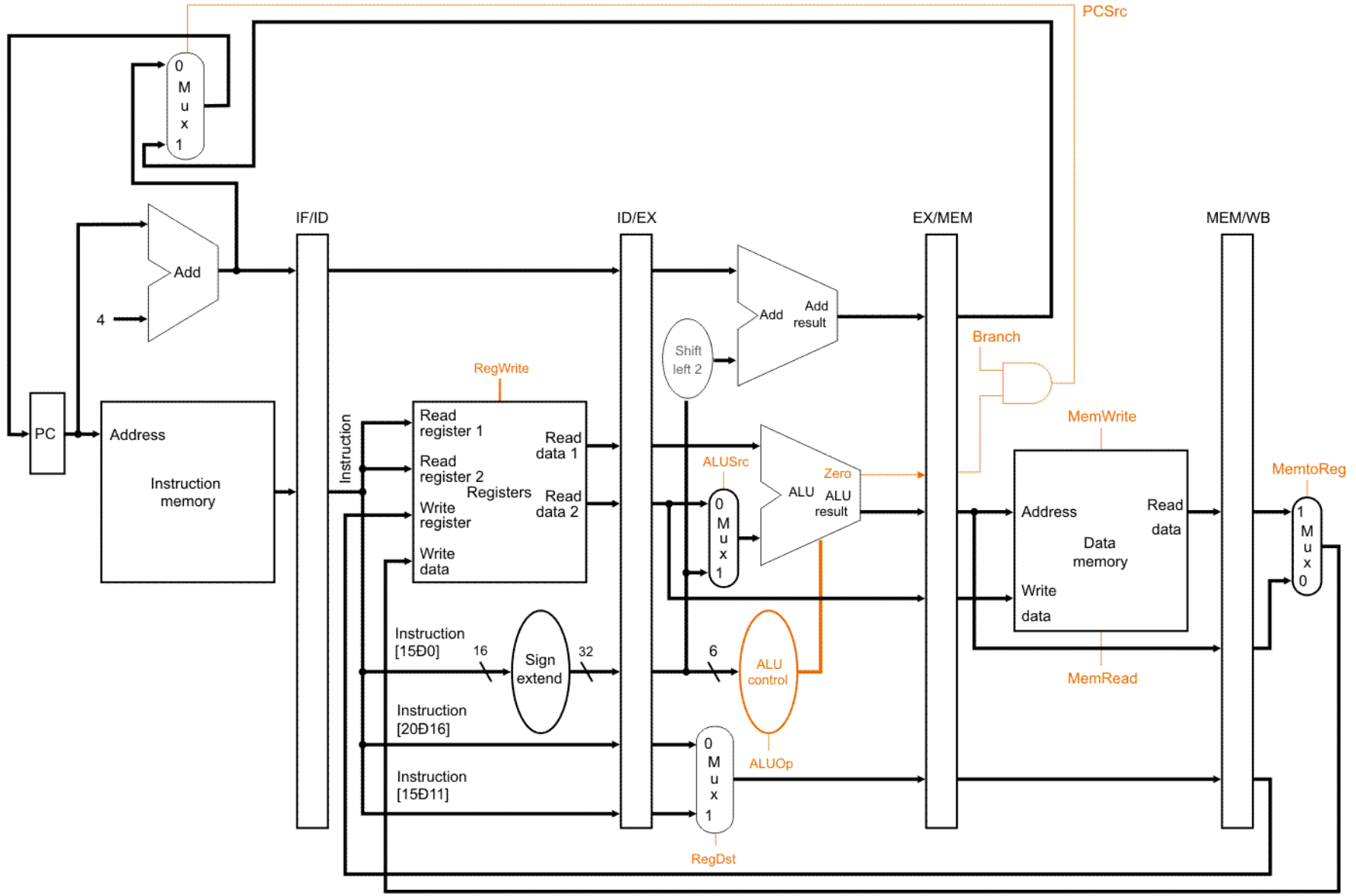
LW / SUB (5)


sub R2, R3, R11
lw R10, 20(R1)
Memory
Write back
IF/ID
ID/EX
EX/MEM
MEM/WB
M u x
Add
4
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Data memory
Read data
Write data
16
Sign extend
32
Clock 5

LW / SUB (6)



Control Signals in Pipelined Arch.

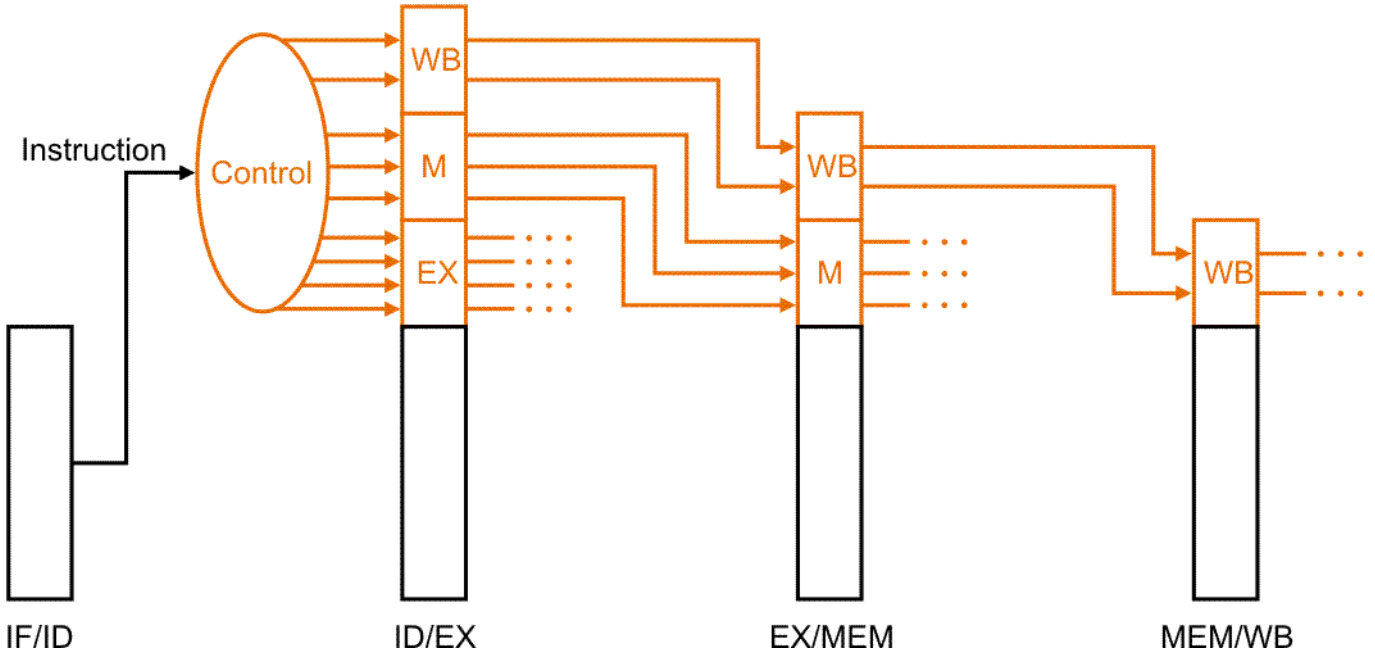


Pipelined Architecture Control

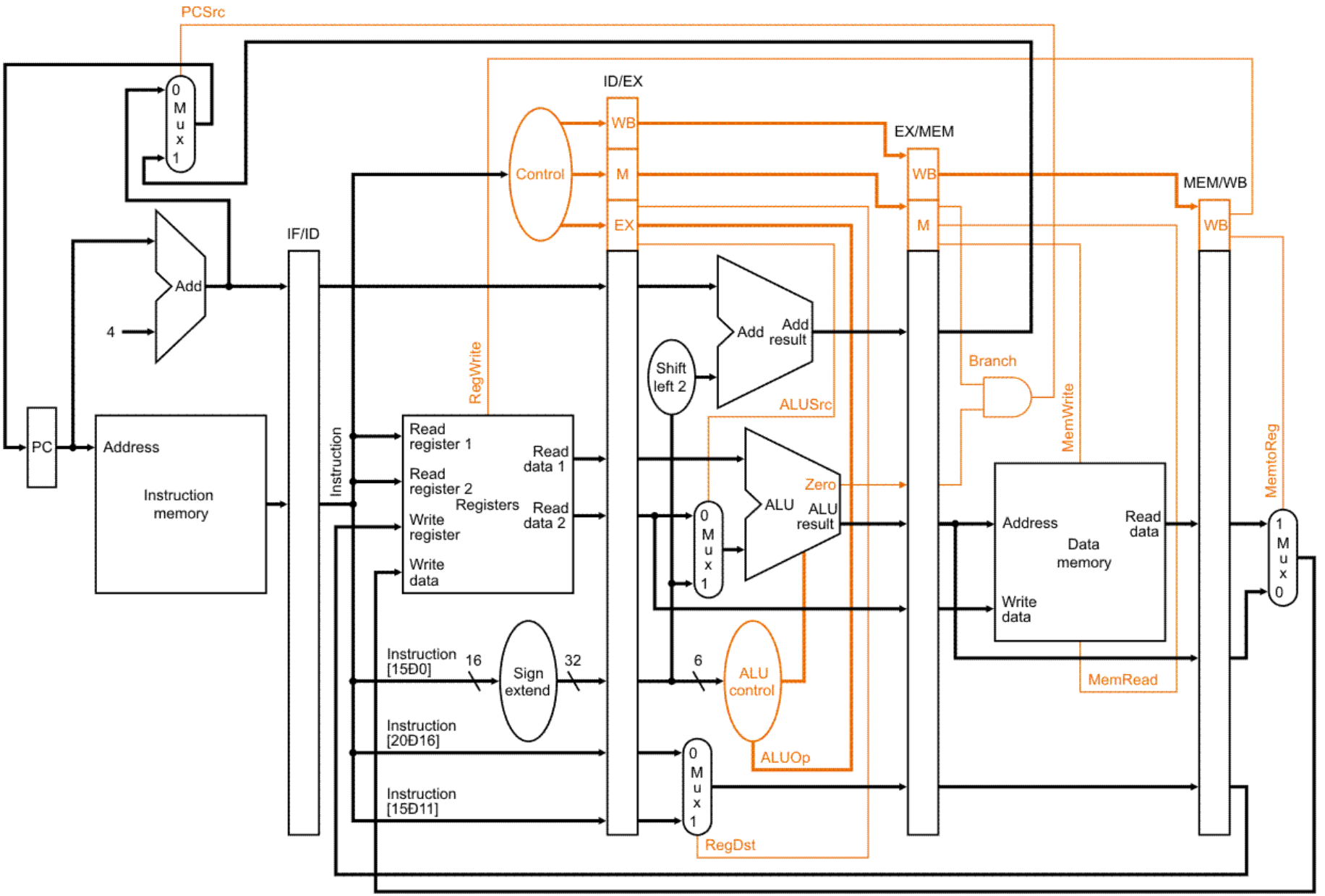
- Control is simple – no reuse of hardware resources
- All control signals must be developed in decode stage
- Control unit implementation is combinatorial – all signals can be described with a truth table (similar to Single-Cycle)
- Control signals must be application must be distributed over several stages of instruction execution (similar to Multi-Cycle)
- Instruction must "carry" the control signals through the pipeline and use them selectively

Control Signals Distribution

	EX			MEM			WB	
	RegDst	ALUop	ALUsrc	Branch	Mem Read	Mem Write	Reg Write	MemTo Reg
R-type	1	10	0	0	0	0	1	0
lw	0	00	1	0	1	0	1	1
sw	X	00	1	0	0	1	0	X
beq	X	01	0	1	0	0	0	X



Minimal Pipelined Architecture

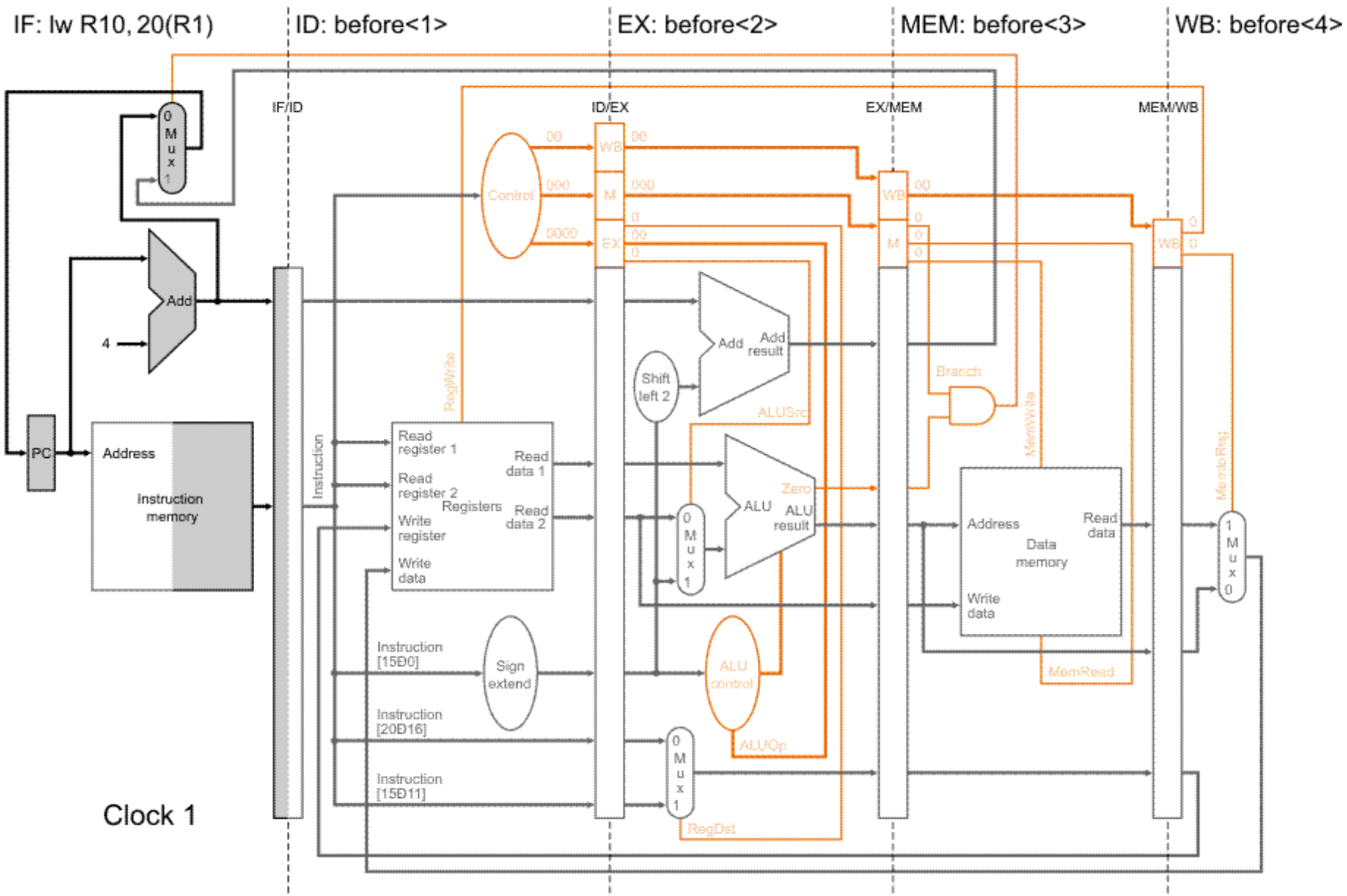


Instructions in a Pipeline – Example

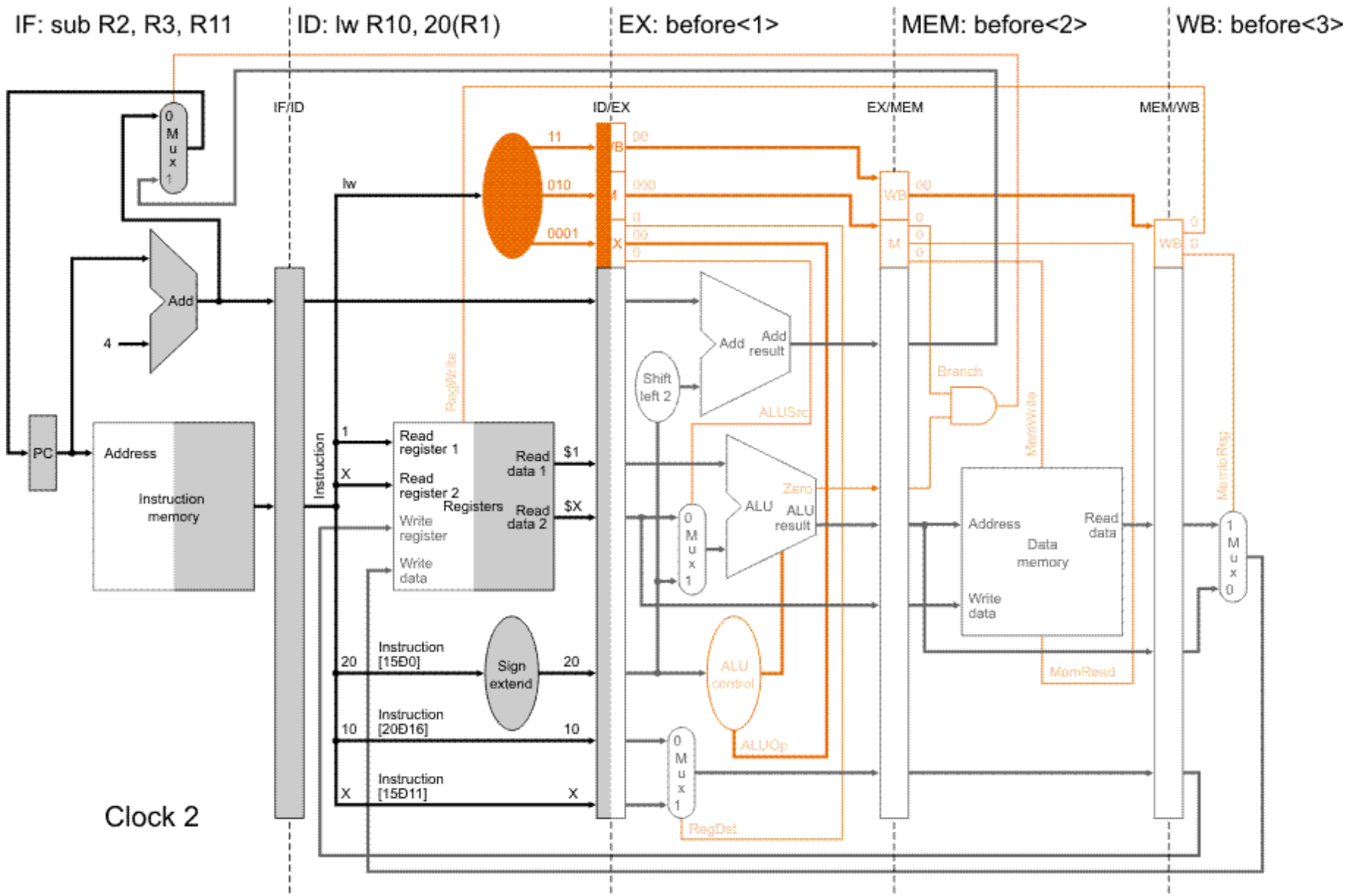
- Important Assumption:
  - no hazards (structural, data, control)

```
lw      R10,20(R1)
sub     R2,R3,R11
and     R4,R5,R12
or      R6,R7,R13
add     R8,R9,R14
```

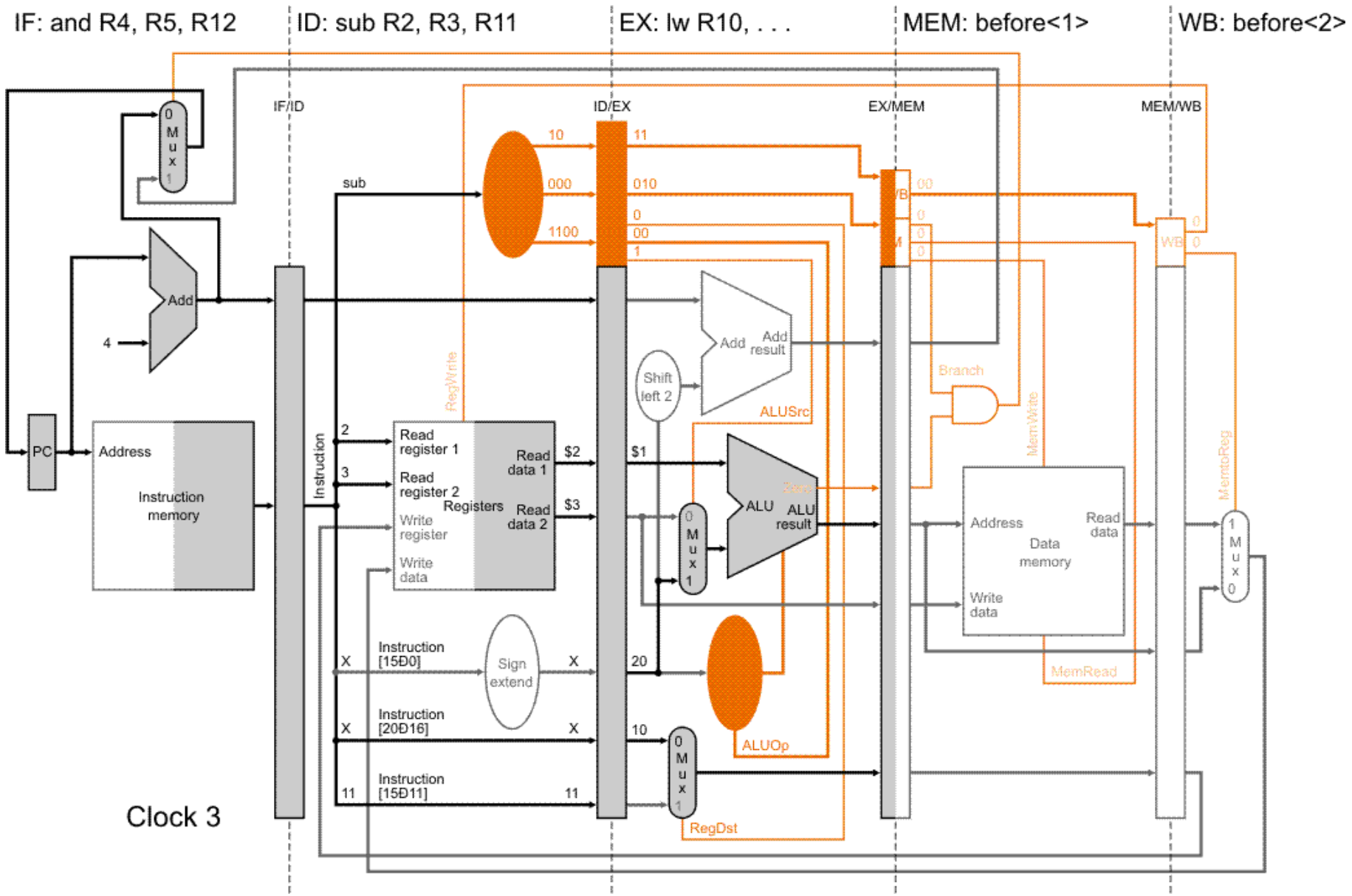
Pipeline (1)


IF: lw R10, 20(R1)
ID: before<1>
EX: before<2>
MEM: before<3>
WB: before<4>
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
PC
Address
Instruction memory
Instruction
Control
WB
M
EX
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
RegWrite
Add
Add result
Shift left 2
ALUSrc
ALU
Zero
ALU result
ALU control
ALUOp
Sign extend
Instruction [15Ð0]
Instruction [20Ð16]
Instruction [15Ð11]
RegDst
Branch
MemWrite
MemRead
MemtoReg
Address
Data memory
Read data
Write data
Clock 1

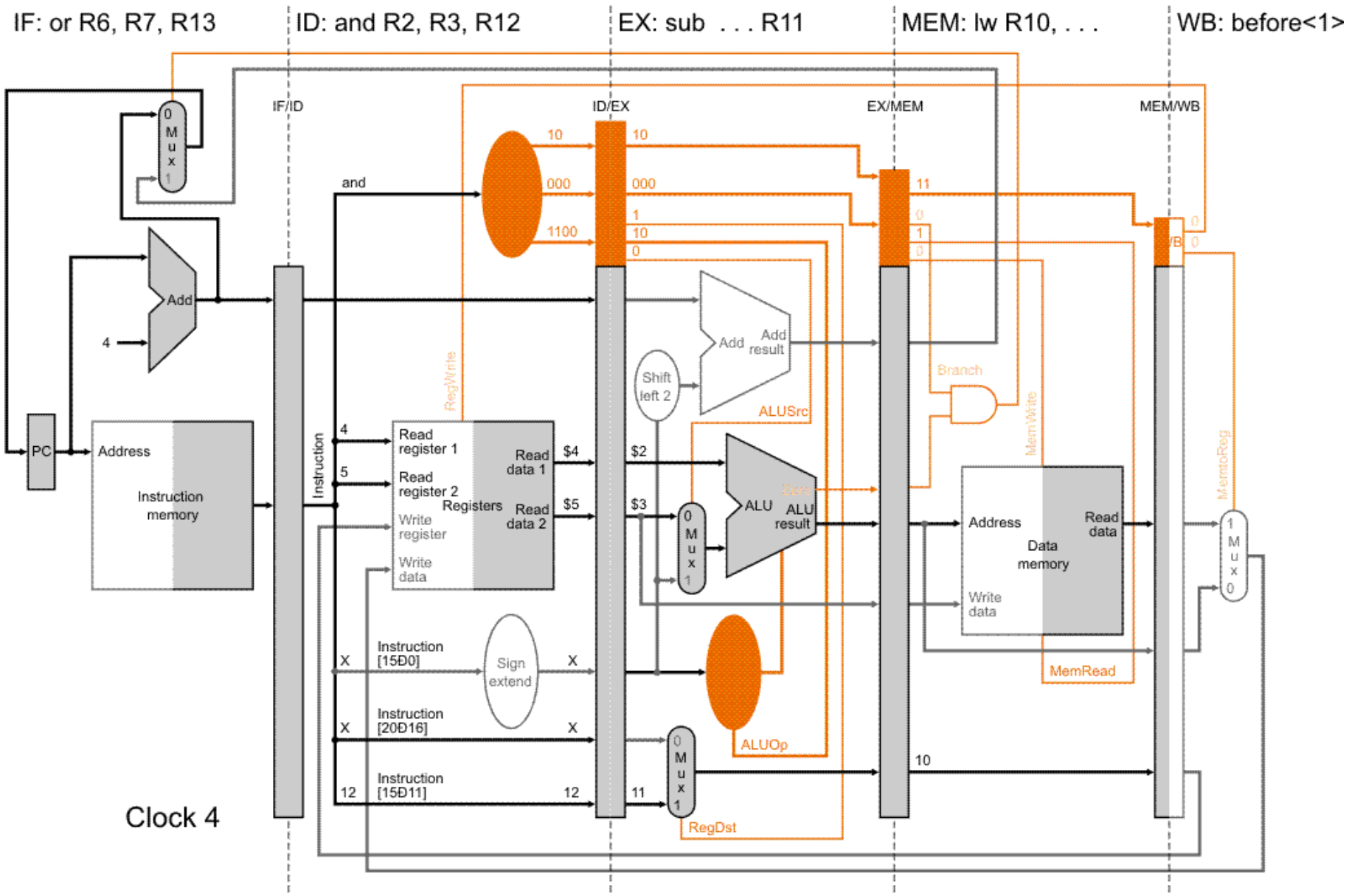
Pipeline (2)


IF: sub R2, R3, R11
ID: lw R10, 20(R1)
EX: before<1>
MEM: before<2>
WB: before<3>
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
RegWrite
lw
11
010
0001
Shift left 2
Add
Add result
ALUSrc
ALU
Zero
ALU result
ALU control
ALUOp
RegDst
Branch
MemWrite
MemRead
MemtoReg
Address
Data memory
Read data
Write data
Instruction [15Đ0]
Instruction [20Đ16]
Instruction [15Đ11]
Sign extend
20
10
Clock 2

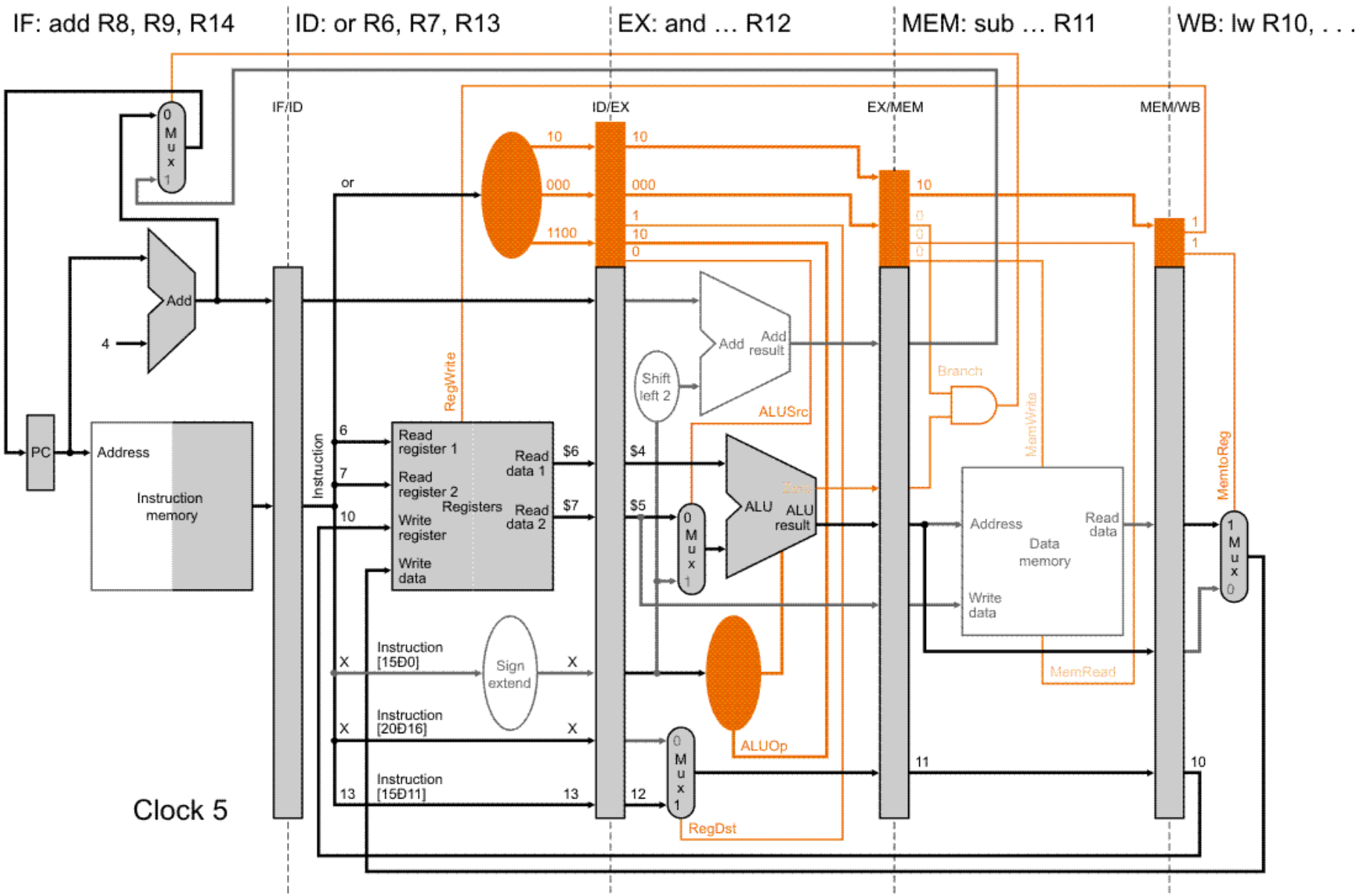
Pipeline (3)


IF: and R4, R5, R12
ID: sub R2, R3, R11
EX: lw R10, . . .
MEM: before<1>
WB: before<2>
IF/ID
ID/EX
EX/MEM
MEM/WB
sub
Add
4
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
RegWrite
Shift left 2
Add
Add result
ALUSrc
ALU
ALU result
Zero
Branch
MemWrite
MemtoReg
Address
Data memory
Read data
Write data
MemRead
Sign extend
Instruction [15Ð0]
Instruction [20Ð16]
Instruction [15Ð11]
ALUOp
RegDst
Clock 3

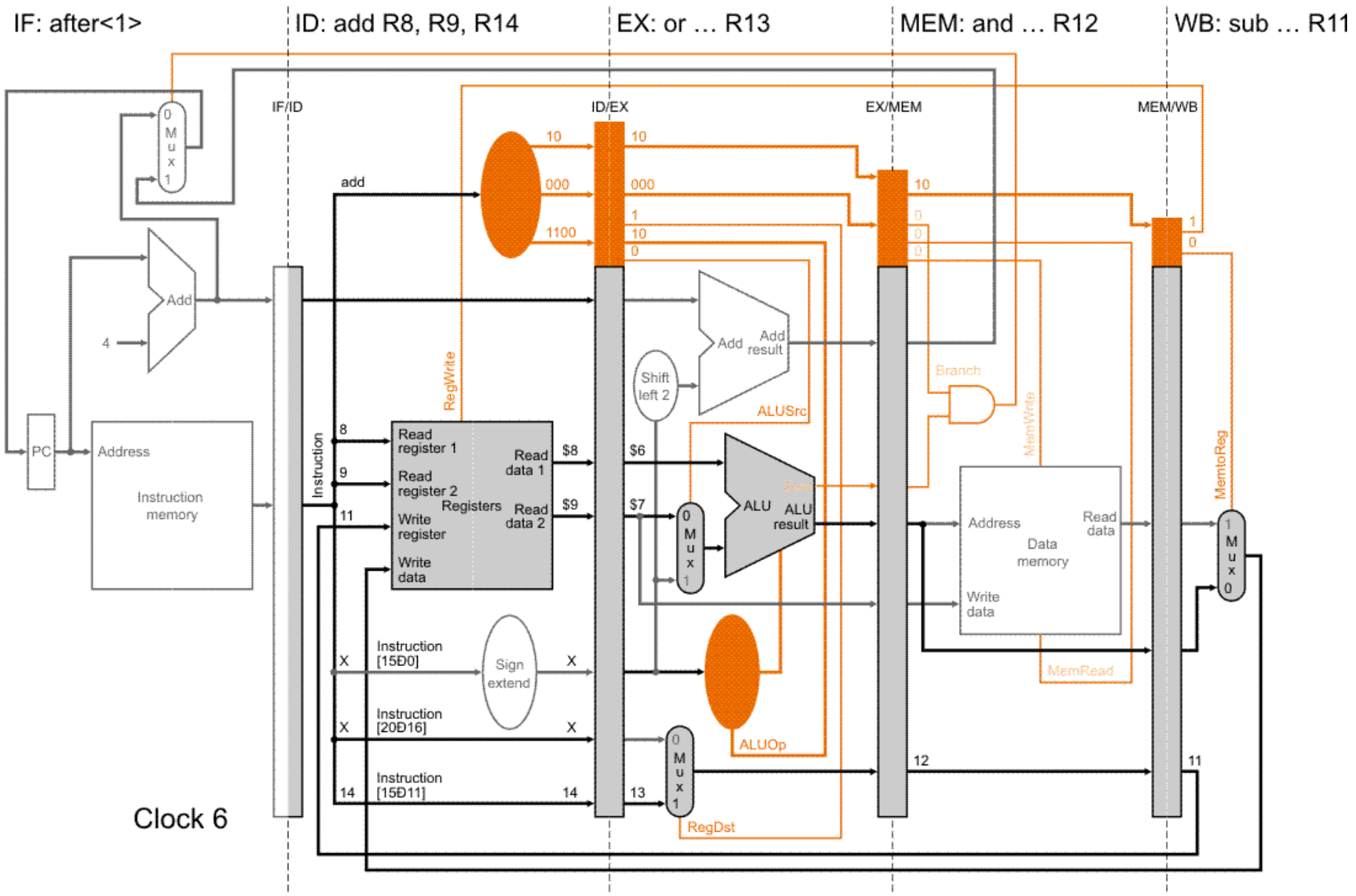
Pipeline (4)


IF: or R6, R7, R13
ID: and R2, R3, R12
EX: sub . . . R11
MEM: lw R10, . . .
WB: before<1>
IF/ID
ID/EX
EX/MEM
MEM/WB
and
Add
4
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
RegWrite
Shift left 2
Add
Add result
ALUSrc
ALU
ALU result
Zero
Branch
MemWrite
MemtoReg
Address
Data memory
Read data
Write data
MemRead
Instruction [15Đ0]
Instruction [20Đ16]
Instruction [15Đ11]
Sign extend
ALUOp
RegDst
Clock 4

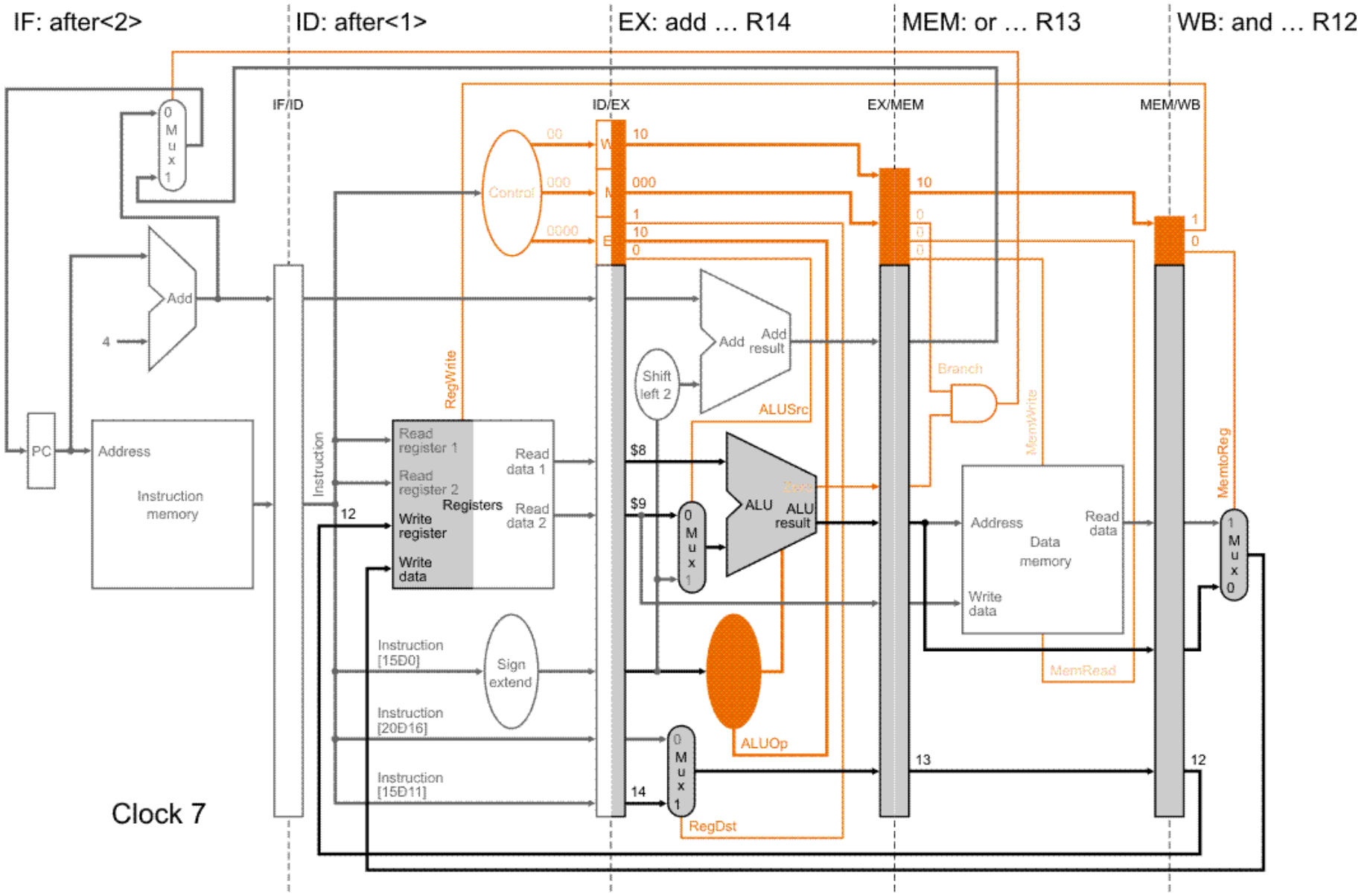
Pipeline (5)



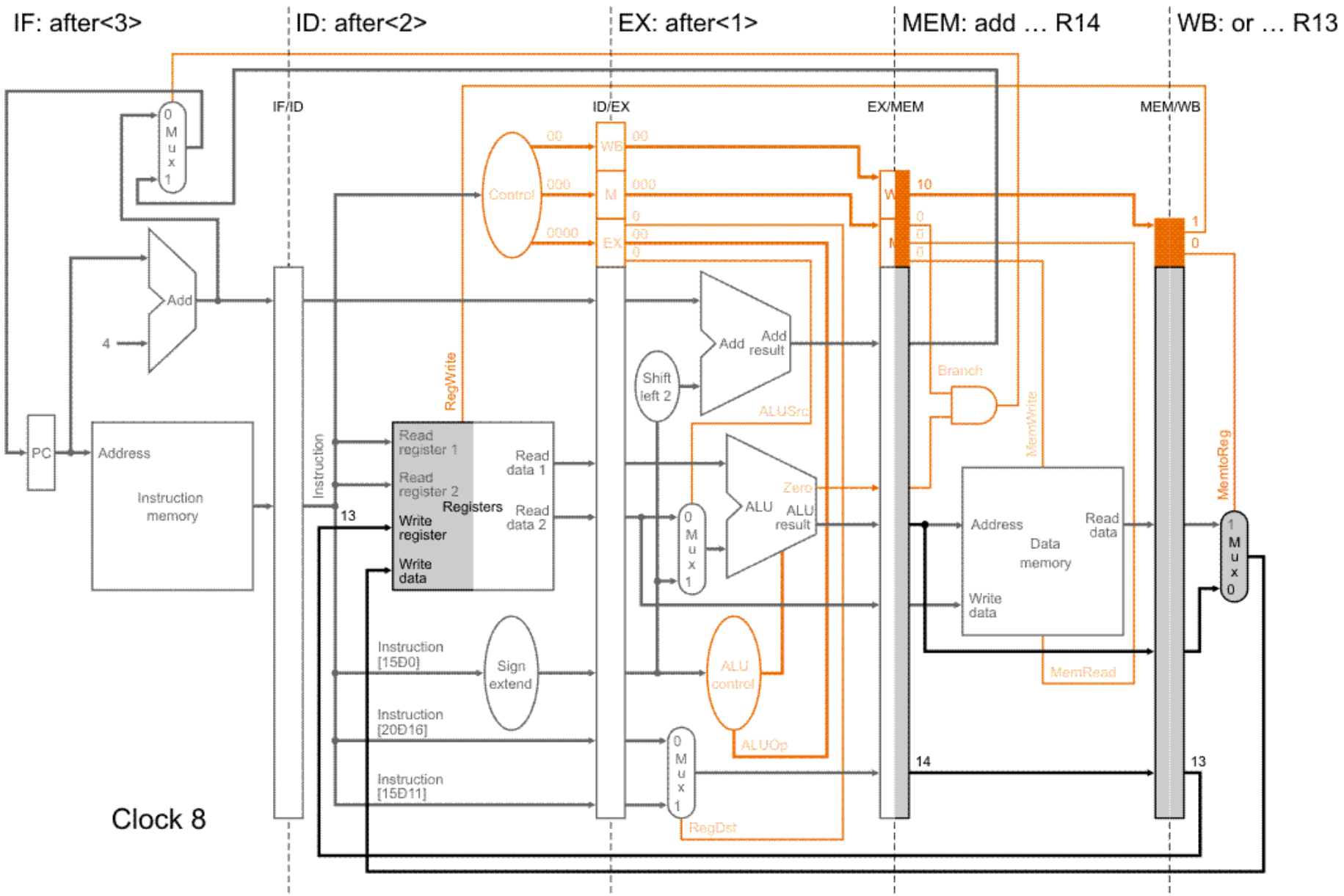
Pipeline (6)


IF: after<1>
ID: add R8, R9, R14
EX: or … R13
MEM: and … R12
WB: sub … R11
IF/ID
ID/EX
EX/MEM
MEM/WB
Clock 6

Pipeline (7)


IF: after<2>
ID: after<1>
EX: add … R14
MEM: or … R13
WB: and … R12
IF/ID
ID/EX
EX/MEM
MEM/WB
Clock 7
PC
Address
Instruction memory
Add
4
Instruction
RegWrite
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
12
Control
00
000
0000
10
000
1
10
0
Add
Add result
Shift left 2
ALUSrc
$8
$9
ALU
ALU result
Zero
Branch
MemWrite
MemRead
MemtoReg
Address
Data memory
Read data
Write data
Instruction [15D0]
Instruction [20D16]
Instruction [15D11]
Sign extend
ALUOp
RegDst
14
13
12

Pipeline (8)



Pipeline (9)